GROUPE RENAULT

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

ETSEIB

Implementation of neural network for autonomous vehicle

Emilien LAURET

Master Thesis Automotive Engineering

Tutors : Maxime Derome, Guillermo Pita-Gil

Speaker : Juan Manuel Moreno Eguilaz

September 6, 2017

Acknowledgements

I would like to thank Maxime Derome and Guillermo Pita-Gil to have given to me the opportunity to discover the new and exciting domain of artificial intelligence, and for all your help and encouragement during the whole project.
A big thanks to the whole Renault data fusion team for offering me a great place to work on my Master Thesis. It was a great pleasure to share lunch and coffee with you.
Thanks to Juan Manuel Moreno Eguilaz for support and managing the relation between Renault and the ETSEIB.

Emilien LAURET

Contents

1 Introduction

1.1 Context

Today, it is impossible not to have heard about autonomous vehicles. Television, magazines, social networks, the vehicle which behaves alone invaded all the media faster than our roads. It is necessary to imagine a vehicle which would allows to drive for the pleasure, and not during boring phases and without interest as in traffic jams or on highways. Laure Pimmel, strategy manager Life one Board Renault Group assures in TECH' OFF n°13, L'aube du véhicule autonome [cd16] " According to a study, 80 % of the interrogated persons imagine themselves in the steering wheel of an autonomous vehicle by ten years ". The driving can be perceived as time lag for the driver. He could then dedicate this time to other activities on board, as checking his e-mails, on-line shopping, watching a movie, reading a book or a simply time of relaxation.

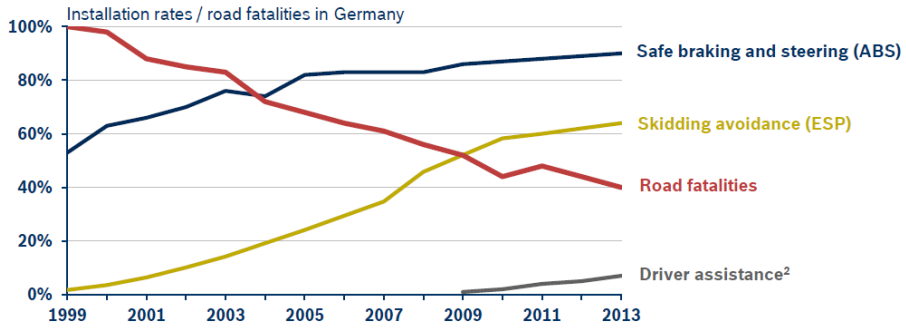


Figure 1: Road safety in Germany - influence of driver assistance [Mes15]

It has been years the car manufacturers work above and anticipate the advent of the self driving, in particular by the progressive development of ADAS (Advanced driver assistance systems), the objective being to reinforce the road safety. We learn for example that the integration of the AEBS (Autonomous Emergency Braking System) or ACC (Adaptive Cruise Control) reduces the number of accidents from 20 to 40% [cd16]. More generally, the integration of safety active systems would have reduced of 60 % the deaths on the German roads these last 14 years as we can see on next figure 1, page 5.

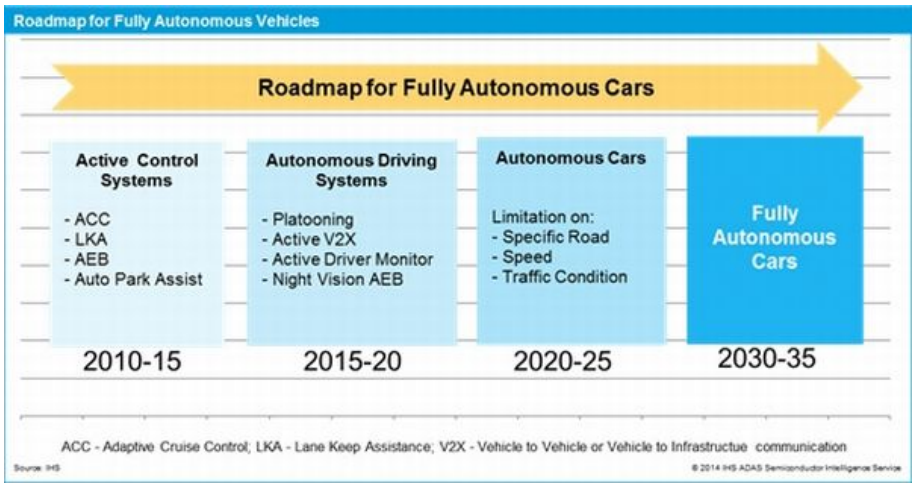


Figure 2: Roadmap for fully autonomous Cars [con15]

In reality, the autonomous vehicle isn't the so natural continuation of ADAS. Of course, the autonomous vehicle re-uses technological blocks from ADAS, but contrary to theses partial automated systems on which the conductor has the decisional power, the autonomous vehicle requires the massive introduction to safety methods, at the same title than nuclear technologies or aerospace.

This self-driving car makes dream as it arouses fear. In any cases, armies of technicians and engineers are working hard on its development, innovation policies encourages it [con15] and big automotive brands promise an arrival in the market in the next few years, with different degrees of automation. Within Renault, CEO Carlos Ghosn said in January 2016 that by 2020, the Renault-Nissan Alliance will launch ten vehicles equipped with different levels of autonomous driving. The fully autonomous function will initially be possible on roads with separate tracks such as motorways, and will progressively extend to more complex situations depending on the experience gained, the maturity of the technology, but also the Regulatory developments.

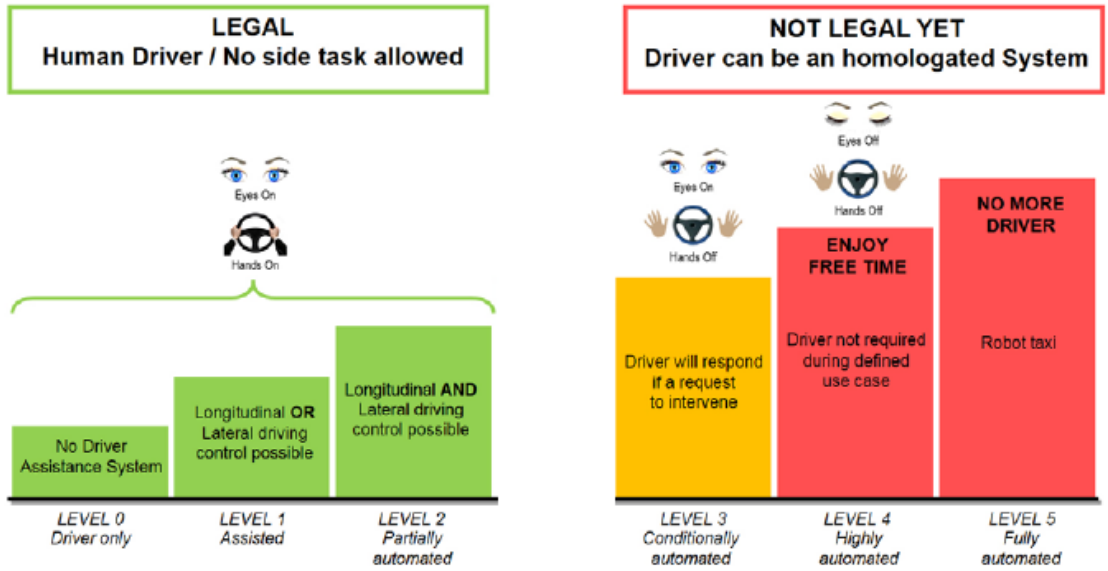


Figure 3: levels of automation and regulation from the SIA [cd16]

Indeed, this revolution in the automotive industry, which is now positioned between the automotive, robot and artificial intelligence domains, involves major legal and regulatory revolutions, with the question that often comes up: "Accident, who is responsible?". However, policies in favour of this technology tend to gradually reduce the regulatory barriers to its development: let us take the example of the opening of roads to the testing of autonomous vehicles under certain conditions [RT.16]: we enter the level 3 of automation.

1.2 Motivations

It's been a long time since planes take off, fly and land in autopilot mode. It could therefore be assumed that the technology of automatic driving is already existing and that it would be sufficient to transfer it from the aeronautical sector to the automotive sector. In reality, the automatic pilot of a plane does not have to take into account very complicated situations: the plane uses a predefined air corridor to be sufficiently distant from the other aircraft. Pedestrian crossing, an obstacle on its way, a two-wheeled vehicle arising laterally, complying with the highway code, break the rules to unblock a situation (crossing a continuous line during an obstacle, forcing the passage to a roundabout, passing an ambulance) ...the autonomous car must be able to make the right decisions on the right time. It is important to realize the scope of the challenge of the autonomous vehicle. Indeed this type of vehicle must perceive, analyse, make decisions and act of itself, in real time, with a very high reliability and for a controlled cost.

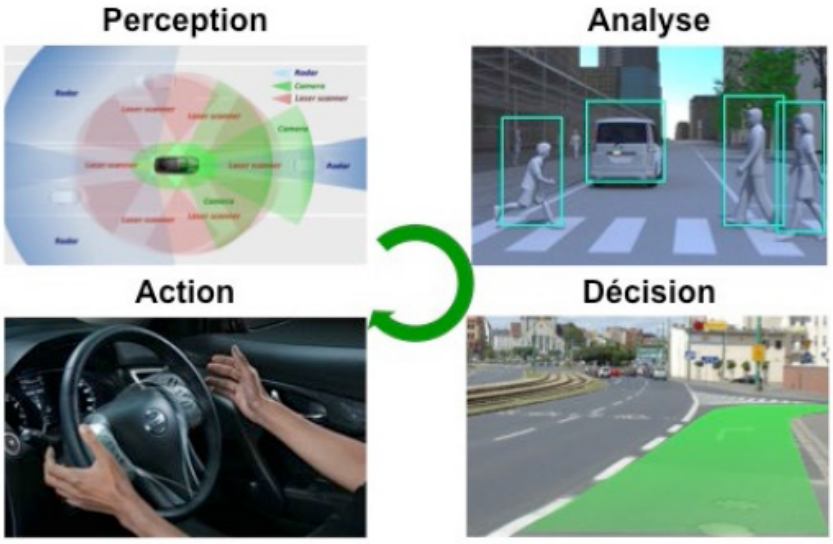


Figure 4: Operating steps in self driving vehicle [cd16]

To be able to perceive the environment in which it evolves as well as the situations presented to it, the vehicle must incorporate a battery of sensors, of different kinds and complementary.

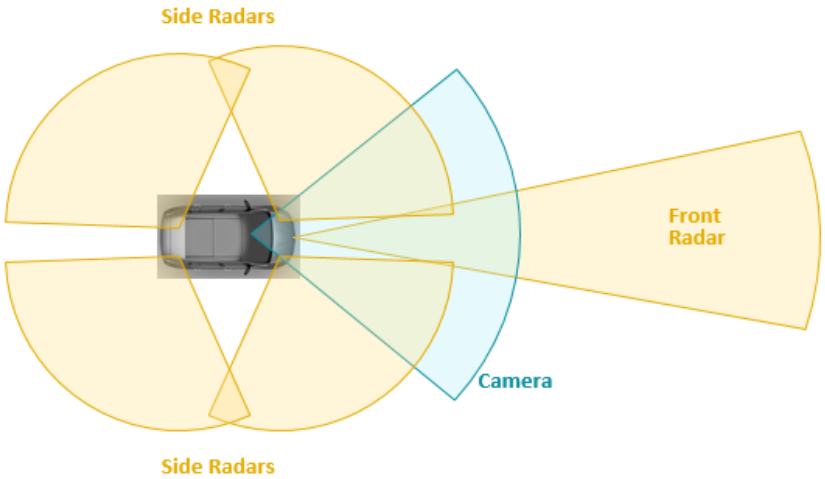


Figure 5: Different kind of sensors in automated vehicle. It can be added the Lidars that are based on a laser technology. Today, this technology is too expensive for the global market.

The radar sees far (200m), measures the distances and speeds of objects, and is robust to weather conditions. It can be positioned at the front of the vehicle to see with some advance a possible obstacle on the road, but also can be positioned on the sides of the vehicle to track vehicles in the vicinity. The problem with radar is that it is not able to differentiate the type of obstacle (motorcycle, infrastructure, stone ...), and then the camera enters the scene. The camera is very interesting for the identification of objects, but has a shorter range (80m), and is subject to weather conditions. A third type is the ultrasonic sensor, which has a range limited to a few meters, but very economical and useful for the detection of nearby obstacles. A fourth family of sensors will emerge in the coming years : the laser scanner, more commonly called LIDARS, seen 6, page 8. This type of sensor scans the whole of the vehicle's turn, allowing to retain the advantages of the laser while allowing a more efficient recognition than the camera. This type of sensor is used on prototypes obtain a ground truth, and evaluate the results obtained with the lower costs camera and radar sensors.

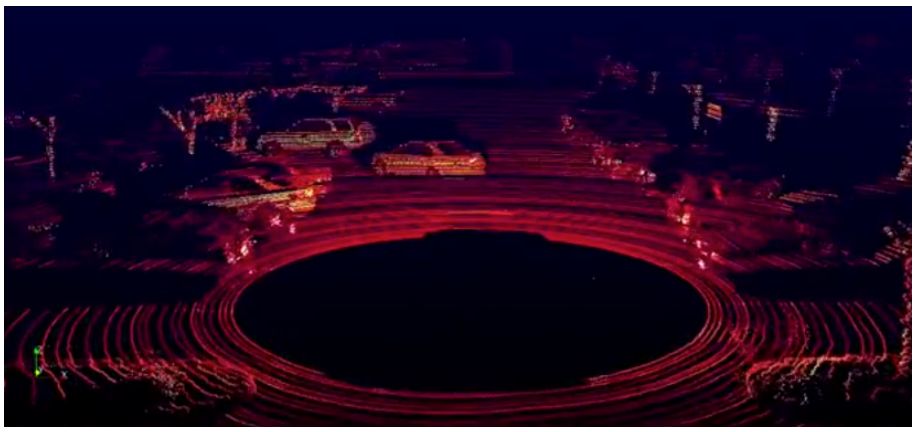


Figure 6: Shot taken from stream of image Velodyne's HDL-64E lidar mounted on a Lexus vehicle

With this whole set of sensors, a big amount of data is obtained.

- The first objective is to construct a model of the environment in which the vehicle evolves. So it is necessary to be able to do a cross-check through the whole sensors data to determine a reliable representation of the surroundings. This amount of data is as heterogeneous as there is different types of sensors. We enter in the field of the sensor data fusion. The matter is to obtain a rich and reliable representation of the environment, ready to be analysed in the next step.

- The second stage is the analysis, which takes into account the planned itinerary, the representation of the environment obtained, integrates a capacity for anticipation by probabilistic means in particular, and defines the objectives to be achieved, taking into account rules of the road and the physical limits imposed by the vehicle's capabilities. In the case of the AEB (Autonomous Emergency Braking) system, it determines the vehicle target to focus on and avoid. At the output of this step, a representation of an environment taking into account a degree of anticipation of the movement of the objects, the objectives to be reached (to follow the road) and the position of the vehicle are obtained. This step can integrate artificial intelligence techniques such as neural networks. This type of tool makes possible to integrate a learning capacity that can be assimilated to the driving experience of a driver that permits him to predict and avoid risked situations.

- The next step is decision. Accelerate, brake, rotate, this step calculates the possible scenarios and selects the decision that best suits the situation and the objective.

- The last step is the one of the action, which receives the orders of the decision step, and which applies them through laws of commands controlling the actuators.

This project deals about the analysis block. The subject is to propose a method that provides a predictive information of cut-in occurrence. We want to integrate to the vehicle the possibility

to anticipate this particularly risked scenario. It has to read the environment data built from the fusion block and to give the prediction as an analysis block output.

2 Which algorithm ?

2.1 The objectives

2.1.1 The problem

I was welcomed into the Renault team developing the perception and analysis algorithms for future autonomous vehicles and ADAS. The mission given to me was to explore and propose an algorithm that permits to anticipate the trajectories changes (cut-in) of the vehicles around our ego vehicle.

As human conductors, we have a capacity to anticipate an other vehicle's change of lane watching the scene in the surroundings of our vehicle. From the main context, regarding the vehicle positions, the road, the behaviour of the vehicles, we are able to assess from our experiment of the road whether a situation will occur or not. The problem is to integrate an ability to take into account all the variables of interest in the decision-making chain of the autonomous vehicle, in order to predict a change of lane situation, that can introduce collision risks. Thus, a pre-processed set of sensor data (lasers, cameras), giving a representation of the environment surrounding the vehicle, as well as information on other vehicles (position, speed, acceleration, distance to the center line) would be in input. It would be necessary to obtain at the output of our algorithm, whether or not a user with a dangerous driving is approaching. This change of state will then be taken into account by the decision cell of the autonomous vehicle, which will adapt its decision to ensure increased safety despite the risky behaviour of the other user. More generally, it is necessary to be able to classify situations into several groups, two for example: no risky behaviour detected / risky behaviour detected. The algorithm we want to define is thus in the family of classification methods. Indeed, this algorithms family takes into account certain particular features derived from the input data and then classifies the input sets related to different situations. This family is used, for example, in the bank, for decision support, or even for medical diagnosis, taking into account the symptoms of the patient.

The first objective is to determine the type of algorithms best suited to satisfy the desired function. Then, it is necessary to understand more precisely the functioning of the selected algorithm, its variants. The implementation of these algorithms will be done through the application of concrete examples.

2.1.2 The constraints

In this project, we will limit ourselves to an algorithm proposal and a first level implementation, without taking into account an integration in the vehicle. This second part will be elaborated by a specialized team, working on code optimization, and implementation. Nevertheless, it may be interesting, in the context of the autonomous vehicle, to define the framework of the constraints that are applied to this algorithm. First, in the automotive field, autonomous or not, the robustness of the algorithms is paramount. Indeed, the algorithm can in this case continue to work well in a noisy environment or during unexpected inputs without inducing parasitic results leading to undesired operations. Secondly, of course, in a decision-making application moving a vehicle on the roadway, the real-time aspect is paramount. It is not possible to accept latency of calculation or other temporal delay to the detection and therefore to the decision-making. Take into account the evolution of the situation must be as rapid as possible, to bring about a new and adequate decision. We have seen that the autonomous vehicle involves a massive introduction of safety procedures in the automotive. In this sense, redundancy would be required for the software and hardware implementation of critical electronics.

2.2 Law based algorithms

We have seen that we must look for methods of classification. A first method of classification is the method to the rule-based approach, defined by experts. The aim is to convert expert knowledge and / or experience in the form of applicable rules to discriminate input sets. The difficulty in our case is to be able to define all the rules defining the example class. In effect, it's very difficult to be able to determine rules that predict vehicle behaviour and their interaction depending on road context. This kind of algorithm can be written under a decision tree, and it can seem limited when the number of parameters to take into account comes high. It can be inferred that this kind of algorithm is not adapted to gives a kind of "road experience" to the decision cell of the autonomous vehicle.

2.3 Learning algorithms

2.3.1 What is learning ?

Learning is the process in which different classes of data are discriminated referring to common specificities that characterize the class they belong. Basically, learning is the attempt to define a link between events and consequences with a cause and effect principle. In machine learning, we take data, we train the model on that data, and we want to predict the class of new data. The finality is that the model is tuned during the learning phase, and then it should makes the classification of unfamiliar data. A learning algorithm adapts, step by step, some parameters to improve its performances : it's the learning process. In our case, and in view of the difficulty of extraction of rules in our situation, we turn to learning methods. We can distinguish several types of learning:

- Supervised learning : The algorithm learns to classify the data according to a predetermined discrimination model using examples. The first step in using this type of algorithm is to define a model for labelling examples according to some specific characteristics defined manually. The second step is that of prediction, in which the algorithm will classify situations from the experience gained by learning the labelled examples of the first step.

- Unsupervised learning : This is the case where we do not have labelled examples. The algorithm hasn't an example of classification to learn. The algorithm must find out how to classify the data. This type of method is used when the nature and number of groups to be classified is unknown. It has particularly clustering applications to classify data, or even when it is desired to elicit explanatory hypotheses, and correlate clinical diagnoses for example.

- Semi-supervised learning : A set of indexed and non-indexed examples is available. This method is between supervised and unsupervised learning. It is presented to the algorithm a combination of indexed and non-indexed examples, which makes possible the algorithm to adapt its classification method. The first step of the algorithm is to learn the indexed examples. The algorithm is then used to classify non-indexed examples. Data labelled with a high degree of confidence are included in the indexed examples. And the cycle is repeated until a certain stopping criterion is reached. The labelling of the data is often done manually, it's an human who determines the class of the data in order to obtain the labelled examples. In large datasets, manual indexing can quickly become tedious. Therefore, this technique has an obvious practical aspect, reducing the number of manually labelled examples needed.

- Strengthening learning : The algorithm learns behaviour through observation feedback (or cause-effect). A practical case may be a mobile robot equipped only with a shock sensor. This robot, in an environment, will collide with obstacles and will, by repetition, ultimately learn an optimal trajectory and make a path avoiding obstacles.

Back on our work. The main idea is to classify manually or automatically a set of real vehicle data in various classes : when we are able to see regarding the data (images, vehicles positions, speeds, position to the center of lane) that a specific scenario of our interest is occurring, we label the corresponding data on the corresponding class. We do that for a consequent set of data (for one example of data taken in t time, we index it to a class in which it belongs). This is the constitution of our database. This is the process through the human put his knowledge and capability to classify data. Rules based algorithms can be used to constitute the database more easily if we have access to additional ground reality data, as line overlap. In our case, we determine an algorithm that gives the class of the example regarding the vehicle area overtaking a threshold (25%) in the ego vehicle lane.

Once the database build, it would be presented to learning methods which would be able to adapt its parameters to reproduce the same classification than the labelling of the database examples and learn the way to classify the data from specific data. The interest of learning methods appears when we apply new examples, never seen before. The algorithm is able to give us some classification scoring. It gives us a classification scoring of a non labelled example regarding the learned database composed of 'human' labelled examples. So we are in the domain of supervised learning.

Regarding the very high number of data from which extract the classification (position, speeds, accelerations for each vehicle presents around the vehicle), the adaptive methods seems to be interesting. As its name suggest, this category of algorithms determines a way to classify data adapting its parameters. This adaptation of parameters modify the "behaviour" of the classifier. Because of this characteristic, it opens the possibility to modify theses parameters before the classification process, or even on-line, during the classification process. However this kind of classification is done under a "black box process", which can be crippling for some applications necessarily requiring a physical interpretation of the classification process. Two classes of adaptive methods commonly used are presented here:

- Genetic algorithms : Their goal is to approach a solution of an optimization problem in a relative short time compared to others methods. We try to estimate the function by a natural selection process.The solution is approached modifying parameters by successive iterations, favouring some parameters which induce better optimisation but keeping completing the parameter collection with hazardous parameters values. The main objective is to obtain in a restricted time a relative good optimization where the optimum would need more calculation time.

- Neural networks : One another approach is to take a tree architecture that recalls the law based rules algorithms, and adapt parameters which determine the transfer function of the neural network. It's equivalent to say that the modification of parameters adapts the classification 'laws' of our tree. One more time, theses parameters are without clear physical meaning, and the classification 'laws' of the neural network are only numerical weights and bias determining the output of the classifier. This last method introduces non linearities and can extract features to classify with a high level of abstraction, as the human intuition that a vehicle will do a cut-in.

3 Neural Networks

The main idea of this part is to present neural networks, and provide the general theory used on this master thesis project.

3.1 Global description

Artificial neural networks are inspired by the biological neural networks presents in the human brain. Originally, it is an attempt to modeling the work of the human brain.An artificial neural network is a classification method designed to work on the same (simplified) principle than the human brain works.

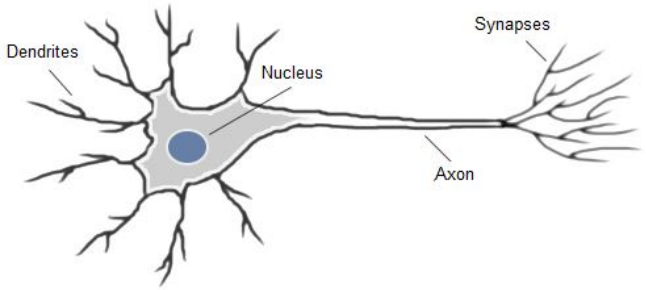


Figure 7: A biological neuron is composed by dendrites that collect inputs signals from the others neurons, a nucleus that introduces a threshold and gives a certain transfer function to the neuron, an axon that transmits the signal and some synapses that distribute the output signal to the dendrites of others neurons.[Jac13]

That leads to a very different concept that how one usually compute code. The human brain data process is done using an enormous amount of little unities : neurons. Theses neurons are connected to lot of others neurons, creating a very important network. Each of these unities collects one input signal (electric signal) from all others cells linked to its input. If the addition of all this instantaneous signals reaches a certain threshold, the neuron emits from its output a signal which is collected by the neurons taken its output in input.

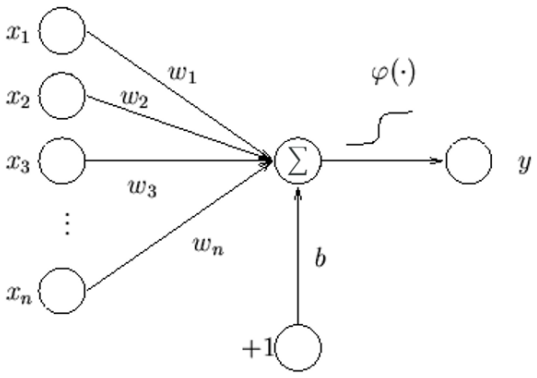


Figure 8: A graphical representation of a simple perceptron. Here y is the output signal, φ is the activation function, n is the number of connections to the perceptron, wi is the weight associated with the ith connection and xi is the value of the ith connection. The b in the figure represents the threshold. Picture by the author of [HON01].

When writing an artificial neural network, this is mimicked by using a "perceptron" as the basic unit instead of the neuron. The perceptron can take several weighted inputs and summarize them, and if the combined input exceeds a threshold it will activate and send an output. The output sent is determined by the activation function and is often chosen to be between 0 and 1 or -1 and 1. Since the derivative of the activation function is often used in the training of the network,

it is convenient if the derivative can be expressed in terms of the original function value, as few additional computations are needed to calculate the derivative in this case. Two applications of this kind of algorithm will be seen in this report. The sigmoid function as activation function φ was chosen for the first application, and for the second the φ is the hyperbolic tangent function. The explanations will be seen on their respective parts. The equation for a perceptron can be written as

$$y = \varphi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

where y is the output signal, φ is the activation function, n is the number of connections to the perceptron, w_i is the weight associated with the ith connection and x_i is the value of the ith connection. b represents the threshold. A graphical representation can be found in figure 8, page 12. The threshold b is associated to a constant input equal to 1. By allowing the network to modify b, a dynamic threshold is achieved. In the case of a perceptron with a sigmoid activation function, the main transfer equation is:

$$u_1 = w_1 * x + b_1$$

$$y = sigmoid(u_1) = \frac{1}{1 + e^{-u_1}}$$

A schematic explanation of the perceptron obtained is proposed on the next figure 9, page 13. Tuning the weight and bias of the neuron, it's searched to approach the desired function. Tuning the two parameters permits the perceptron transfer function to change. While modifying the ratio $\frac{b_1}{w_1}$ changes the threshold value separating the two classes (class 0 and class 1), the modification of the w_1 parameter modify the slope of the transfer characteristic (modify the severity of the classification).

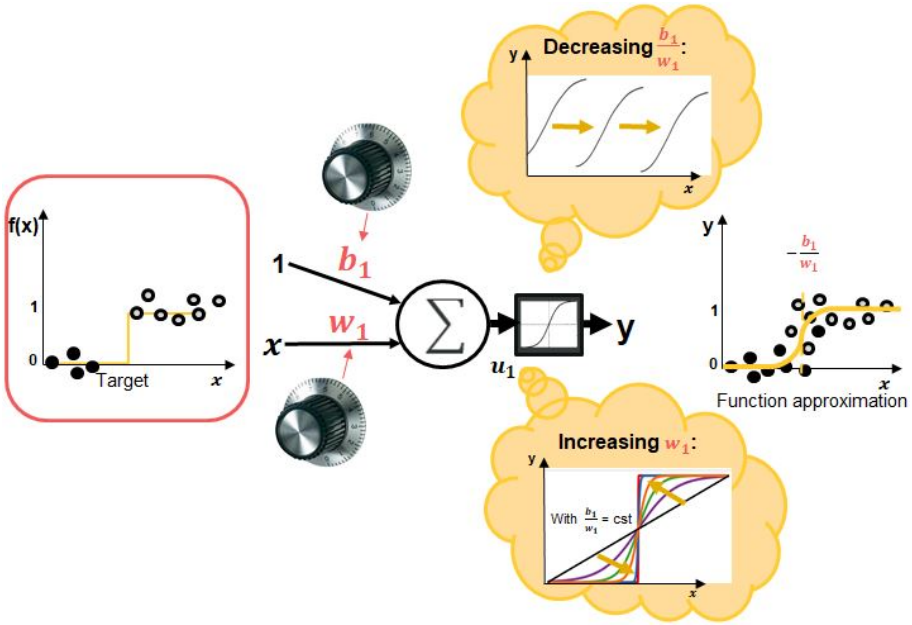


Figure 9: The simplest perceptron (neuron) is composed by one weight W_1 and a bias b_1. Playing with theses two degrees of liberty, the goal is to be able to approximate the target function.

Finally, an artificial neuron can be compared with a parametrizable electronic transistor : when a threshold is reached, the output value change. But in artificial neuron model, the threshold level and its shape is tunable. However it can be asked : What is the interest using this kind of approximation instead of using an other method ? The perceptron introduces naturally a non linearity because of the nature of its activation function, that gives to this method goods characteristics for non linear classification problems. On that way, and dependently of the tuning of the parameters, the neuron is able to classify the inputs with more or less strict fronter between two classes. The simple perceptron shown on the last figure can see its number of inputs increased.

If one input is added, the equations are :

$$u_1 = w_1 * x_1 + w_2 * x_2 + b_1$$

$$y = sigmoid(u_1) = \frac{1}{1 + e^{-u_1}}$$

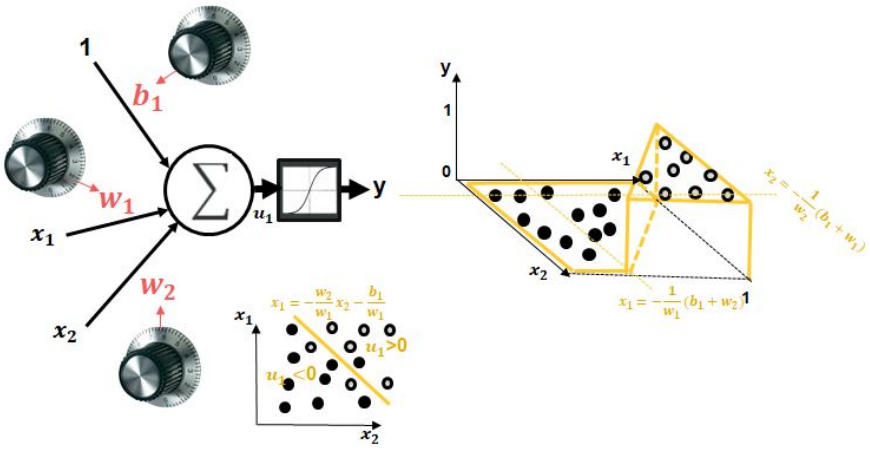


Figure 10: Adding an input x_2, an associated weight w_2 is added. Now the perceptron does no longer separate classes in a 1D space but in a 2D space.

At each input added in the neuron, the classification space dimension increase. However, with one perceptron, the characteristic of the classificator keeps a form similar to the activation function shape. In effect the fronter of classification is only a deformation of the activation function characteristic induced by the tuning of the parameters. That is why it is proposed to build a network of neurons : a simple perceptron is a very simple design, but its strength can be shown when several perceptrons are forming a network. The perceptrons are often organized in layers, where each layer takes input from the previous, applies weights and then propagate signals to the next layer. On the next figure 11, page 15 is proposed a graphical explanation of the interest of the networking of various unitary perceptrons. The first layer is composed of two simple neurons of one input as it was seen figure 9, page 13. Then, their outputs are taken as input of the next layer composed by a two input neuron. This neuron composes the output layer of this three neurons network, and works similarly as presented on the figure 10, page 14. By combining these primary unities, it can be obtained approximations of more complex functions, as more complex form of classifier's fronter. The equation for the following neural network:

$$u_1 = w_1 * x_1 + b_1$$

$$u_2 = w_2 * x_2 + b_2$$

$$o_1 = sigmoid(u_1) = \frac{1}{1 + e^{-u_1}}$$

$$o_2 = sigmoid(u_2) = \frac{1}{1 + e^{-u_2}}$$

$$u_3 = w_3 * o_1 + w_4 * o_2 + b_3$$

$$y = sigmoid(u_3) = \frac{1}{1 + e^{-u_3}}$$

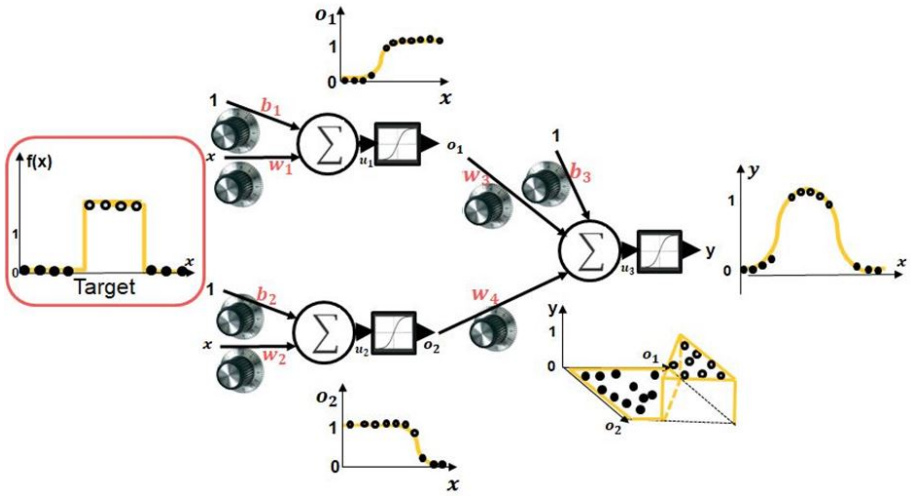


Figure 11: The networking of various simply calculations permits to obtain more complex transfer function.Neural network with one input, two neurons in the hidden layer, and one neuron in the output layer.

Adding more neurons per layer permits the neural network to obtain more expressiveness an permits the neural network to be able to approximate more complex functions. Adding more layers (Deep learning) is a method to go further in the abstraction. A graphical representation of a four layers neural network can be see 12, page 15.

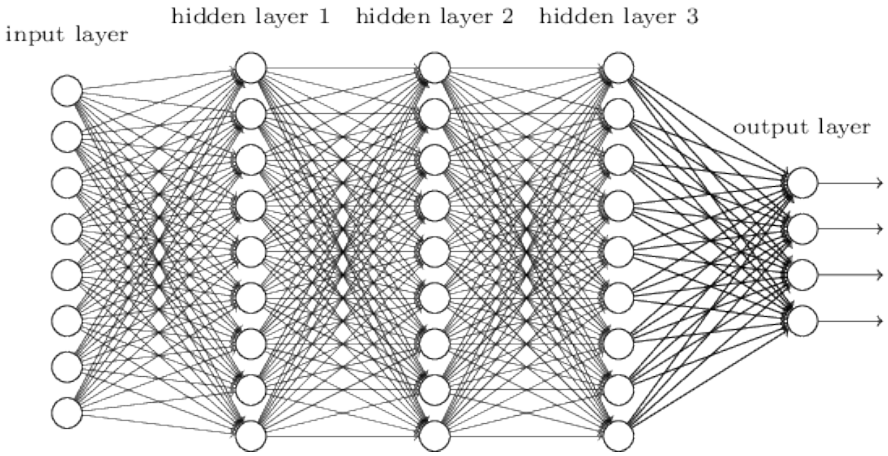


Figure 12: A graphical representation of a neural network composed of three hidden layer, and the output layer.

A learning algorithm must be able to learn from examples and adapt. In an artificial neural network, this is achieved by updating the connections' weights between the neurons. The next part deal with the method to tune theses parameters.

3.2 The learning phase

Train the neural network consists concretely in tuning the weights and bias to reduce a calculated error between the output we want for an associated input and the output obtained. Regarding the dimensions of a neural network, we cannot tune all the parameters manually. This is where the learning algorithm come into play. There is to define a learning algorithm which adapt the weights and bias.

3.2.1 Principle

The main principle of the learning phase is the following : We have a database composed by couples of inputs and associated desired outputs of theses inputs. There is to define fixed weights and bias that permits to approach the most the desired outputs for each inputs of the entire database. It is a general optimization problem. There is to reduce a general error between the outputs calculated on a certain configuration and the target desired. The learning process is composed of the following steps :

1. Weights and bias are initialized more or less randomly.
2. The input vector is propagated onto the neural network, it's called the Front Propagation. It's obtained a certain output vector.
3. This output vector calculated is compared with the target output vector desired for this input vector, and the error made by the network is calculated.
4. From this error, weights and bias are updated according to a chosen method, this is the Back Propagation.

   Then this process loops in the three last steps until the method has reached a stop criterion. This criterion can be a maximum number of iterations, or a classification score.

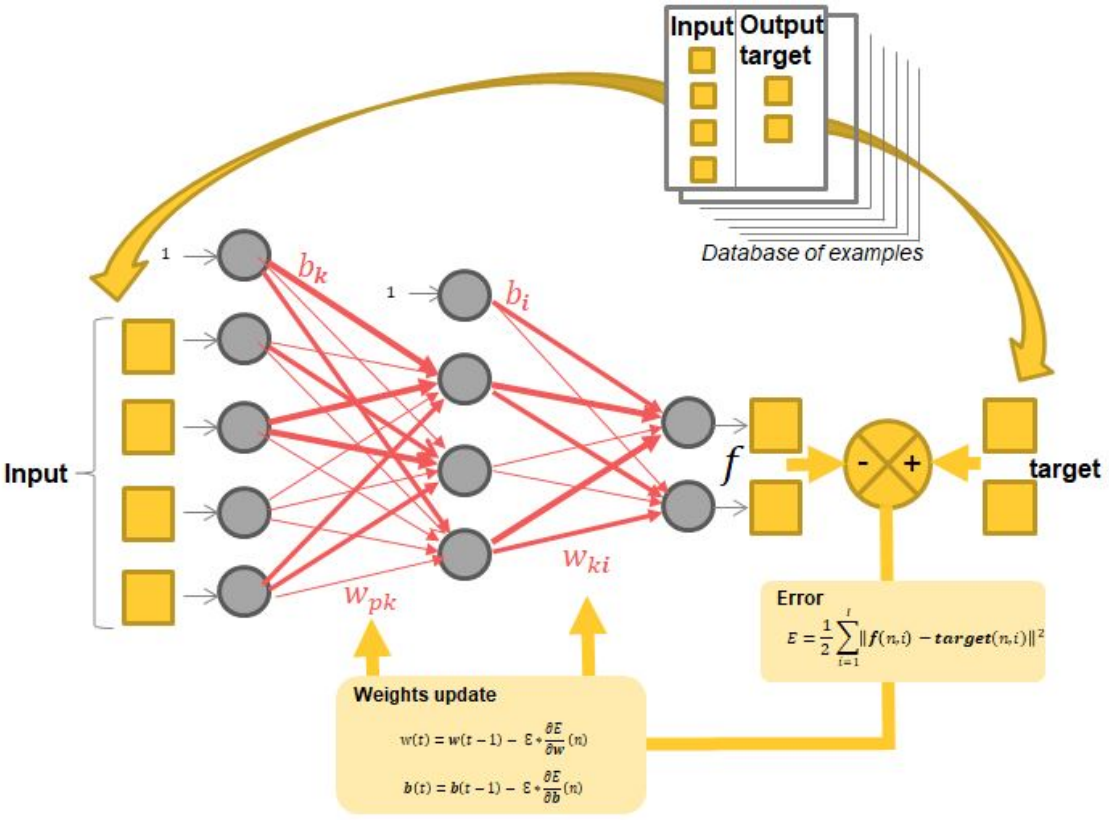


Figure 13: Graphical representation of the gradient descent learning process.

3.2.2 The choice of the learning algorithm method

We will focus on the most used learning process in neural network, the gradient descent based algorithms.
There are tree variant of the gradient descent algorithm. The total batch gradient, the stochastic gradient and the mini-batch gradient. The first one consist in calculating at each iteration the mean of the error gradient on the total training database examples and then modifying the weights and bias. So the parameters are modified only once after calculation of error for all the training database. It can seem very long before obtaining a significant modification of weights and so

a learning process very long. And in practice it is, if there is no a high optimization of the calculation (parallelization, GPU implementation..). So it is not a recommended method for very large databases. The good advantage is to converge slowly but surely to the global minimum. The figure 14, page 17 is an attempt to graphically represents the total batch gradient method.

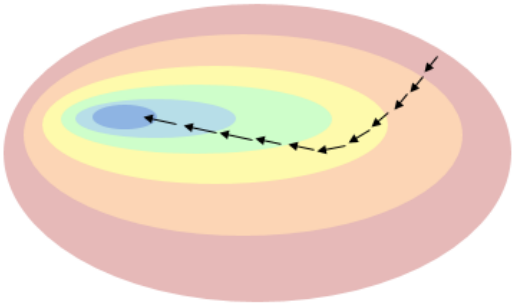


Figure 14: Graphical representation of total batch gradient optimization process

The opposite method is the most used by learning machine users : the stochastic gradient variant. Here the idea is to take randomly one example into the training base, calculate his error gradient and modify weights and bias. This process is reiterated until obtaining convergence near the global minimum. The update of parameters is done after each example calculation instead of through all the database, that induce much faster convergence that the previous method. The global minimum cannot be reached but actually in practice, the goal is to sufficiently approach the minimum and not reach the exact minimum. The figure 15, page 17 is an attempt to graphically represent the stochastic gradient method.

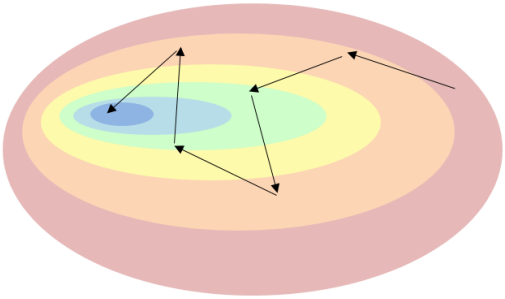


Figure 15: Graphical representation of stochastic gradient optimization process

The third variant, the mini-batch gradient, consists in taking the positive points of the two previous methods. We form a batch of n examples (20 to 100 approximately) taken randomly, we calculate the mean gradient error for this batch and after that, we modify the weights and bias. We reiterate this process until obtaining the convergence of the method. The mini-batch gradient can be more efficient than the stochastic gradient method because of the possibility to optimize the regrouped calculation of gradients for the entire batch. The calculation into the batch can be parallelized or putted in matrix calculation form for time optimization. The mean gradient error done on a batch of examples permits a smoother optimization and approach the global minimum. The size n of the batch is determined to have a suitable speed (depending of optimized calculations methods or available hardware) and a precise convergence[Ng11].

In this work is used the stochastic gradient descent method after have implemented and tried the three methods. This method seemed to be faster than others (mini-batch gradient less interesting because of lack of optimization in the mini-batch calculations code). We will see further the theoretical part showing the equations of the method in detail in the next section 3.2.3, page 18.

3.2.3 Theory & Algorithm

This section deals with the presentation of the theory of the developed algorithm. As seen in part 3.2.1, page 16, the learning process is composed of various steps.

- The first step refers to the initialization of the parameters. It will be seen how are initialized the weights and bias for each of the two specific applications seen in part 6, page 24 for the first and part 11, page 47 for the second.
- Then, the second step of the algorithm is the front propagation. It's simply the calculation through the mathematical definition of the neural network.

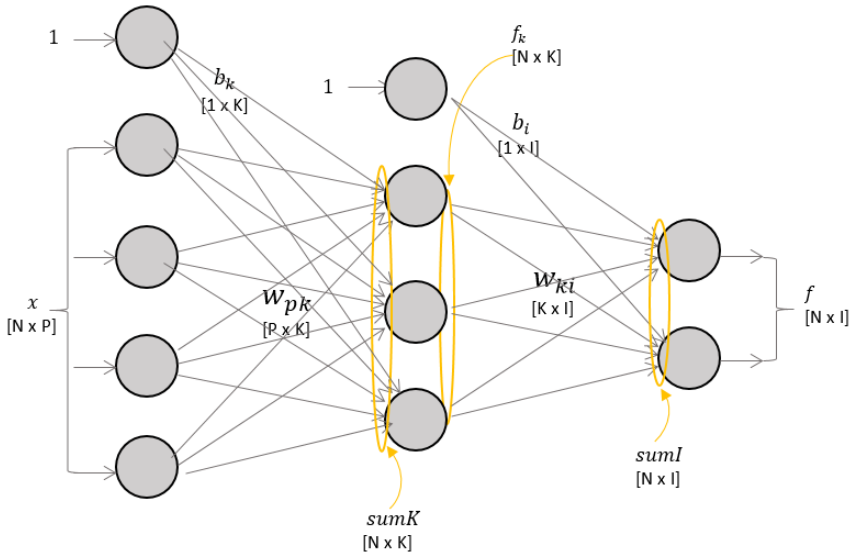


Figure 16: The graphical representation of the generic two layer perceptron architecture implemented (one hidden layer and one output layer), independently of the number of input P, number of hidden neurons K and number of outputs I.

$$N : number\ of\ examples\ presented$$
$$P : dimension\ of\ input\ vector$$
$$K : number\ of\ hidden\ neurons$$
$$I : dimension\ of\ output\ vector$$

The following are the general equations for the two layer perceptron referring to the figure 16, page 18:
Front propagation equations :

$$sumK(n,k) = \sum_{p=1}^{P}[w_{pk} * x(n,p)] + b_k \tag{1}$$

$$f_k(n,k) = \varphi(sumK(n,k)) \tag{2}$$

$$sumI(n,i) = \sum_{k=1}^{K}[w_{ki} * f_k(n,k)] + b_i \tag{3}$$

$$f(n,i) = \varphi(sumI(n,i)) \tag{4}$$

With φ the activation function.

The third step of the learning phase is the calculation of the error obtained referred to the output calculated f on the previous step of front propagation and the output desired a:

$$E = \frac{1}{2}\sum_{i=1}^{I} ||f(n,i) - a(n,i)||^2 \tag{5}$$

From this error assessment, the error is propagated through each network connexion, an on each layers, from the output layer to the input layer. It's the back propagation or retro propagation process. It is reminded the derivatives of the two activation functions used in this paper :

$$for\ \varphi(x) = sigm(x),\ \varphi' = \varphi(1-\varphi)$$

$$so\ f(n,i)' = f(n,i)(1-f(n,i)) \quad and \quad f_k(n,k)' = f_k(n,k)(1-f_k(n,k))$$

$$for\ \varphi(x) = tanh(x),\ \varphi' = 1-\varphi^2$$

$$so\ f(n,i)' = 1-f(n,i)^2 \quad and \quad f_k(n,k)' = 1-f_k(n,k)^2$$

The derivatives forms are simple, this is why theses specifics functions are chosen to be activations functions for neural network.

Retro propagation equations :

$$\frac{\delta E}{\delta w_{ki}}(n) = \frac{\delta E}{\delta f(n,i)}.\frac{\delta f(n,i)}{\delta sumI(i)}.\frac{\delta sumI(i)}{\delta w_{ki}} \tag{6}$$

$$\frac{\delta E}{\delta w_{ki}}(n) = (f(n,i) - a(n,i)) * f(n,i)' * f_k(n,k) \tag{7}$$

$$\frac{\delta E}{\delta b_i}(n) = \frac{E}{\delta f(n,i)}.\frac{\delta f(n,i)}{\delta sumI(i)}.\frac{\delta sumI(i)}{\delta b_i} \tag{8}$$

$$\delta E/(\delta b_i)(n) = (f(n,i) - a(n,i)) * f(n,i)' \tag{9}$$

$$\frac{\delta E}{\delta w_p k}(n) = \frac{\delta E}{\delta f(n,i)}.\frac{\delta f(n,i)}{\delta sumI(i)}.\frac{\delta sumI(i)}{\delta f_k(n,k)}.\frac{\delta f_k(n,k)}{\delta sumK(k)}.\frac{\delta sumK(k)}{\delta w_{pk}} \tag{10}$$

$$\frac{\delta E}{\delta w_p k}(n) = \sum_{i=1}^{I}[(f(n,i) - a(n,i))f(n,i)' * w_{ki}] * f_k(n,k)' * x(n,p) \tag{11}$$

$$\frac{\delta E}{\delta b_k}(n) = \frac{\delta E}{\delta f(n,i)}.\frac{\delta f(n,i)}{\delta sumI(i)}.\frac{\delta sumI(i)}{\delta f_k(n,k)}.\frac{\delta f_k(n,k)}{\delta sumK(k)}.\frac{\delta sumK(k)}{\delta b_k} \tag{12}$$

$$\frac{\delta E}{\delta b_k}(n) = \sum_{i=1}^{I}[(f(n,i) - a(n,i))f(n,i)' * w_{ki}] * f_k(n,k)' \tag{13}$$

When the error for each connexion is determined, each weights and bias are updated referring to their importance on the error's induction:

$$w_{ki}(t) = w_{ki}(t-1) - \epsilon\frac{\delta E}{\delta w_{ki}}(n) \tag{14}$$

with $\epsilon \in R$

$$b_i(t) = b_i(t-1) - \epsilon\frac{\delta E}{\delta b_i}(n) \tag{15}$$

with $\epsilon \in R$

$$w_{pk}(t) = w_{pk}(t-1) - \epsilon\frac{\delta E}{\delta w_{pk}}(n) \tag{16}$$

with $\epsilon \in R$

$$b_k(t) = b_k(t-1) - \epsilon\frac{\delta E}{\delta b_k}(n) \tag{17}$$

with $\epsilon \in R$

The main difference between the three kind of gradient descent learning algorithm operates in this last step, on the weights and bias update. While the stochastic method chosen here does an update after each loop, the batch gradient method sums the gradients calculated on the whole database, summing at each new loop and then doing the mean. This mean value of the gradient induces a smoother evolution of the error, while our stochastic method will be much more faster.

It's desired to pass in matrix written for calculation optimization. So for an example in input, we put every parameter in vectorial form.

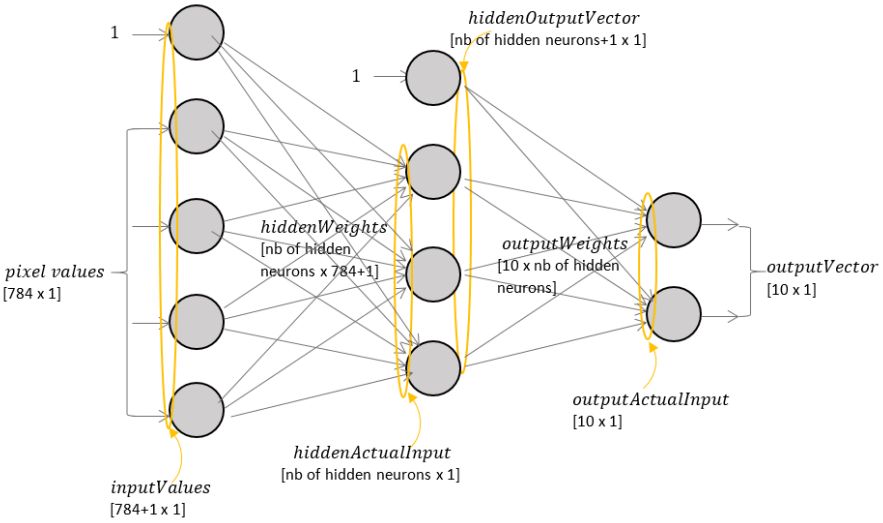


Figure 17: Graphical representation of the matrix form neural network representation.

Front-propagation :

$$inputValues = [1; pixelvalues] \tag{18}$$

$$hiddenActualInput = hiddenWeights * inputValues \tag{19}$$

$$hiddenOutputvector = [1; \varphi(hiddenActualInput)] \tag{20}$$

$$outputActualInput = outputWeights * hiddenOutputVector \tag{21}$$

$$outputvector = \varphi(outputActualInput) \tag{22}$$

Back-propagation :

$$outputDelta = \varphi'(outputActualInput) * (outputVector - targetVector) \tag{23}$$

$$hiddenDelta = \varphi'(hiddenActualInput) * (outputWeights(:, 2 : end) * outputDelta) \tag{24}$$

$$outputWeights = outputWeights - coeff * outputdelta * hiddenOutputvector \tag{25}$$

$$hiddenWeights = hiddenWeights - coeff * hiddenDelta * inputVector \tag{26}$$

3.2.4 Practical considerations

The tuning of the different parameters that characterize the neural network and its learning process will be seen on the first practical application part I. The tendencies regarding to the modification of each parameters will be proposed, to assess the interest of each parameter (number of examples in the database, number of hidden neurons, value of the learning rate).

To make the training more efficient, techniques such as momentum can be used. Momentum adds inertia on the modification of parameters and avoid high error variations. If the amplitude with which are updated the weights and bias is too small the network will take too long to converge, while if it is too large the network might never converge and begin to oscillate instead. When using momentum, the learning rate $coeff$ is calculated dynamically during the run, on the basis that a

weight which is often changed in the same direction most likely should be big, and can be changed faster.

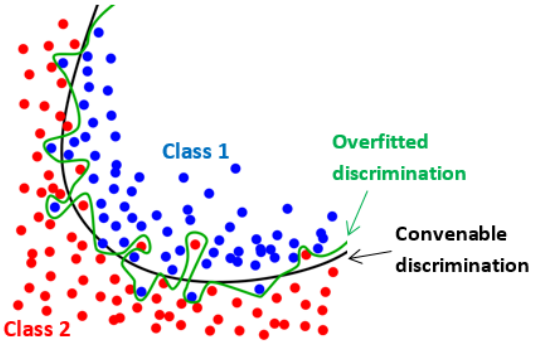


Figure 18: Graphical representation of over-fitting [ICK08]

One problem with the artificial neural network's approach is over-fitting of the data, which happens when the classifier becomes too good at recognizing the training examples, at the expense of not being able to recognize an unknown input. The error is excessively reduced between target and training data and it occurs the rejection of some elements of the test base from the classes they belong. The graphic 18, page 21 illustrates the over fitting phenomenon.

This can be avoided by cross-validation in which the network is trained on one set of data, and then evaluated on a separate one. It permits to detects the start of the phenomenon : when the classification score on the test base start to decrease, the learning phase should be stopped. Example on real first application on 7.4, page 33.

4 Transition

We have seen that pre-defined rule-based methods are not adapted to our problem. We have turned to learning methods. From their predictive capacity, the learning methods allow a certain robustness of the decision set. Indeed, even in the event of a fault on a sensor input, learning should be able to rebuild the signal or simply dispense with it and be able to detect the situation adequately. Due to the type of algorithm chosen, a certain degree of robustness is achieved. Given the number of parameters and the complexity of the problem, we have no particular interest in the fact that the classification procedures are understandable by the user. The choice is therefore open between symbolic and adaptive methods. We can propose an algorithm that maintains a certain capacity for learning on-line, that is to say, capable of continuously learning new cases, such as a driver gaining experience through the experience of relatively new situations. It seems then advisable to move towards the adaptive algorithms. The neural networks would seem to be promising, it imitates the functioning of the human brain, which is capable of performing tasks that appear simple to us like the recognition of faces, grasp an object or predict a cut-in regarding the traffic context but which are extremely difficult tasks for a machine. While a conventional algorithm works with a set of rules and calculations, a neural network algorithm work with images, pictures and concept.

Part I
First application : Handwritten digits recognition

The interest of learning methodologies like neural network is to be able to learn from entire data bases and then be able to classify data. To develop a first neural network algorithm, and be able to train it, and test it, it's needed to have a large database of examples, defined by input vectors and their output target vector associated. We will test the recognition of handwritten digits application, for which a consequent database in free disposition can be found in [YL98]. It contains real-world data and permits to spend minimal efforts on preprocessing and formatting for this first application.

5 Presentation

To define the neural network architecture, a previous study of the data nature is necessary. In effect, there is to define what kind of information it should extract, which are the variables of interest, and adapt the architecture for the specified database. A visualization of samples of the learning database and test database can be seen on figure 19, page 22. It's wanted the neural network to learn on the first database (tune its parameters to correctly classify on the learning database), and then be able to generalize on the test database (obtain a good score in classifying on the unknown test database).

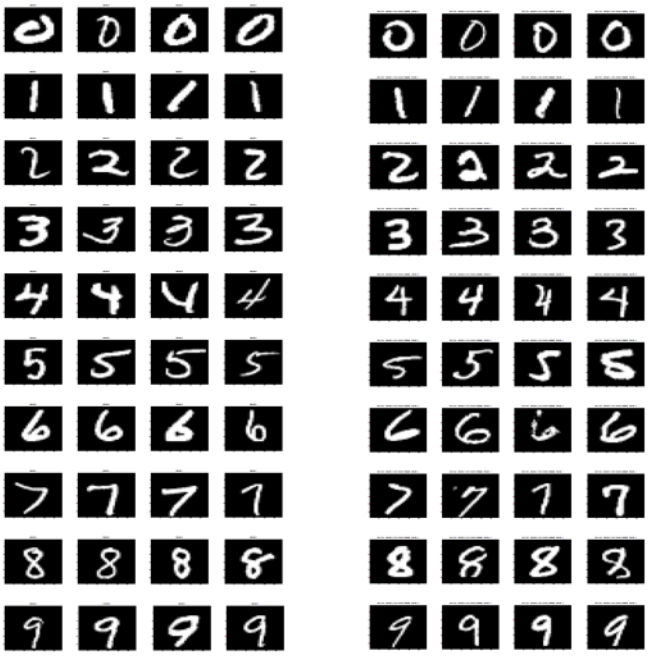


Figure 19: MNIST training database sample (bureau employees' handwritten digits) on the left and MNIST test database sample (high school students' handwritten digits) on the right. Examples taken from the MNIST database available on [YL98]. Pictures of handwritten digits are converted into Grey scale vectors and were manually indexed into the class they should belong

The data files are a set of 10 files, one for each digit from 0 to 9, and each file composed of 1000 examples of an input vector 28x28 pixels and their output vector associated defining their indexation. The input vector is size 28x28 the number of pixel of the digit image, with values in $[0, 1]$ in grey scale, 0 for white and 1 for black. The output vector defines to which class belongs the image digit (class 0, class 1, class 2, ..., 9) in a size 10 vector of value between $[0, 1]$. For

example, in the case of class 2 indexed digit example, the target output vector is [0010000000], and for class 9: [0000000001].

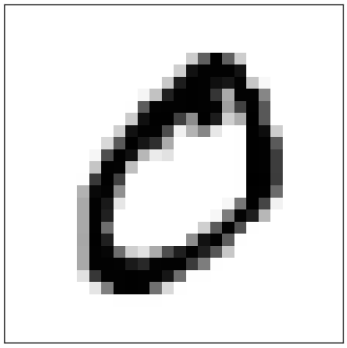


Figure 20: The picture is decomposed in Grey scale values for each pixel. This picture is indexed in class 0, so its target output vector is [1000000000].

With an available database, the second step is to define the neural network architecture.

6 Architecture definition

In this case, inputs and outputs are defined in $[0, 1]$. The information needed to be able to class these pictures is only the degree of darkness of the pixel, so the classification is done regarding the amplitudes of blackness for each pixel. So the activation function that we have chosen is the sigmoid function figure 21, page 23.

$$\varphi(X) = Sigm(X)$$

$$Sigm(X) = \frac{1}{1 - e^{-X}} \tag{27}$$

$$and$$

$$Sigm'(X) = Sigm(X)(1 - Sigm(X)) \tag{28}$$

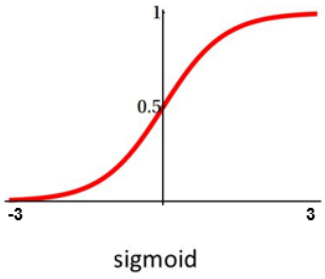


Figure 21: A sigmoid function representation. It takes inputs in $[-\infty, \infty]$ and gives input in $[0, 1]$.

The type of perceptron defined by the choice of its activation function, there is now to define how each perceptrons have to be networked.

For this kind of application we can found in [NIE17] a proposition of architecture which seems working well. This architecture is composed of $P = 28x28 = 784$ inputs referred to the number of pixels of the input picture, I=10 outputs referred to the number of class to discriminate, and K hidden neurons on the hidden layer, with $K = 15$ for a first attempt.

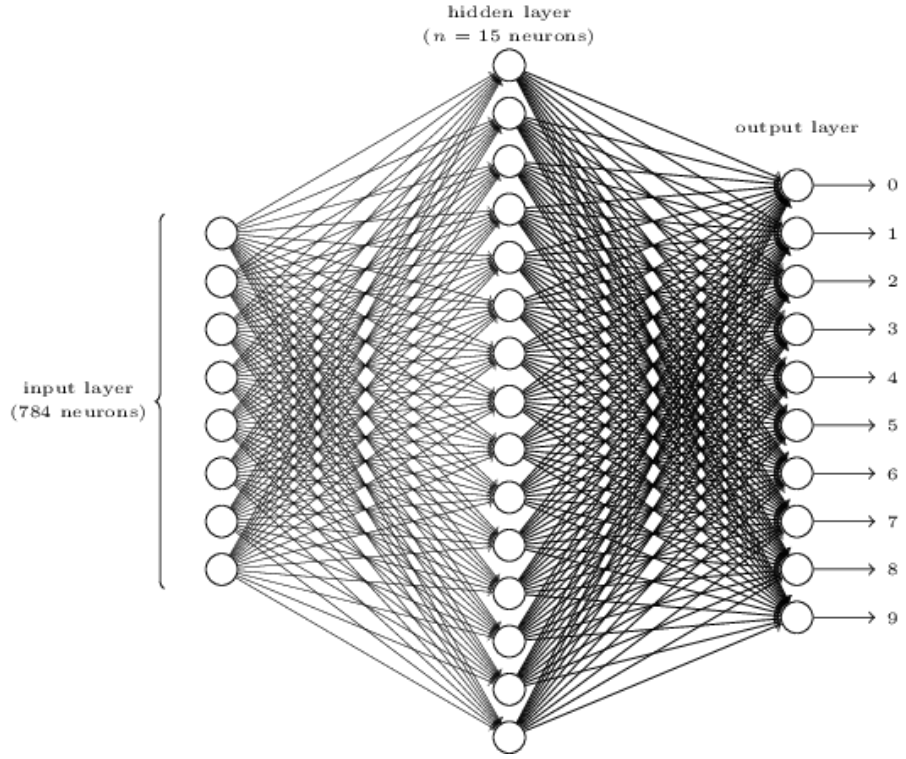


Figure 22: Architecture proposed for handwritten digits recognition problem in [NIE17]

Before launching the learning phase of our algorithm defined in section 3.2.3, page 18 and adapted for this application, there is to define the initialization of the parameters : The initial weights $w_{ki}(0), w_{pk}(0)$ and bias $b_i(0), b_k(0)$ must be chosen. To determinate suitable values, we refer to the paper [XG10] which proposes for a sigmoid activation function value in the following interval:

$$\left[-4 \times \sqrt{\frac{6}{nb_{neuronsIN} + nb_{neuronsOUT}}}, 4 \times \sqrt{\frac{6}{nb_{neuronsIN} + nb_{neuronsOUT}}}\right] \tag{29}$$

7 Algorithm behaviour assessment

7.1 Impact of the database constitution

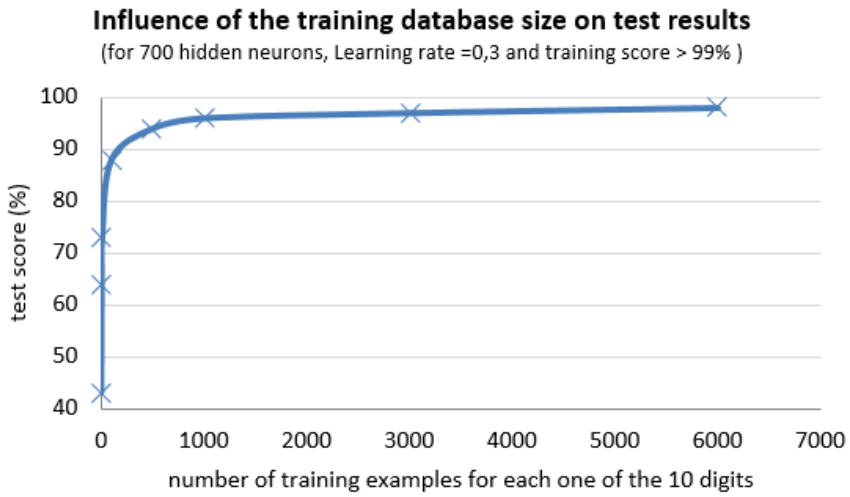


Figure 23: Influence of the training database size on test results

For 1000 training examples in each class, at the point our neural network has learned with a training score of 99%, it is able to be scored at 96% of good classification on test base. It is also full of interest to look at time performances on each case.

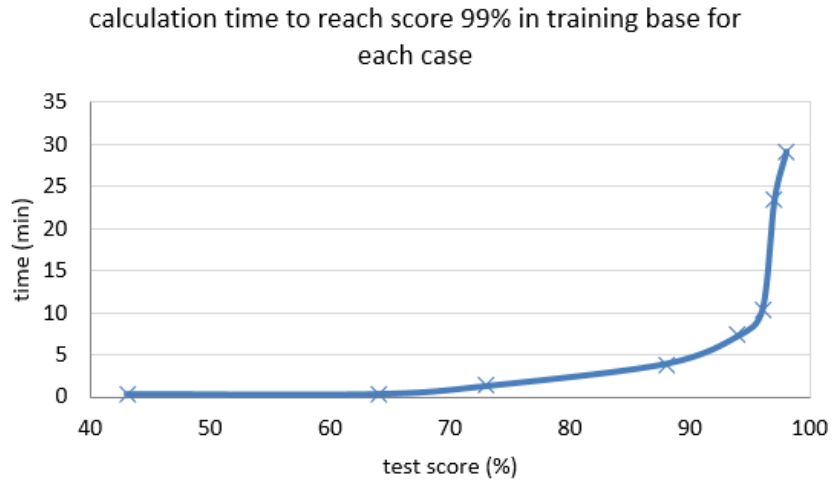


Figure 24: Calculation time to reach 99% in training base for each case

With a learning database composed of 1000 examples for each digits and for a learning score of 99%, it is obtained 96% of good test classification. After the learning base assimilated by the neuron (99% was chosen in place of 100% for time matters), the training score will stay steadily or decrease because of over fitting. In fact to be able to have a significant generalization capability with a minimal training database size, the database needs a variety of examples. We can see that to obtain test score over 96%, the time cost increase dramatically.

In our approach of global determination of the impacts of some principal parameters of our Multi Layer Perceptron, we will continue using 1000 examples per classes, which correspond to the 96% score case and a little over 10 minutes of calculation.

In an attempt to define a convenient iteration number, we can see the next figure which lets us

see that approximately after 100 000 iterations, the test error does no decrease significantly : between iteration 100 000 and iteration 146 000, the test score decreases 1% for 1/3 of iteration and calculation total time, while between iteration 1 and 100 000 it decreases of 95%. We will continue our practical study with 100 000 iteration calculations.

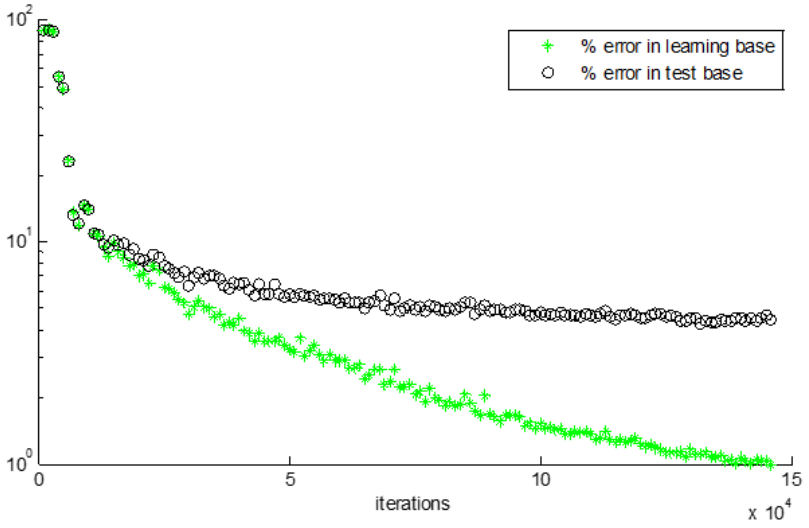


Figure 25: Evolution of classification score on learning database and on test base

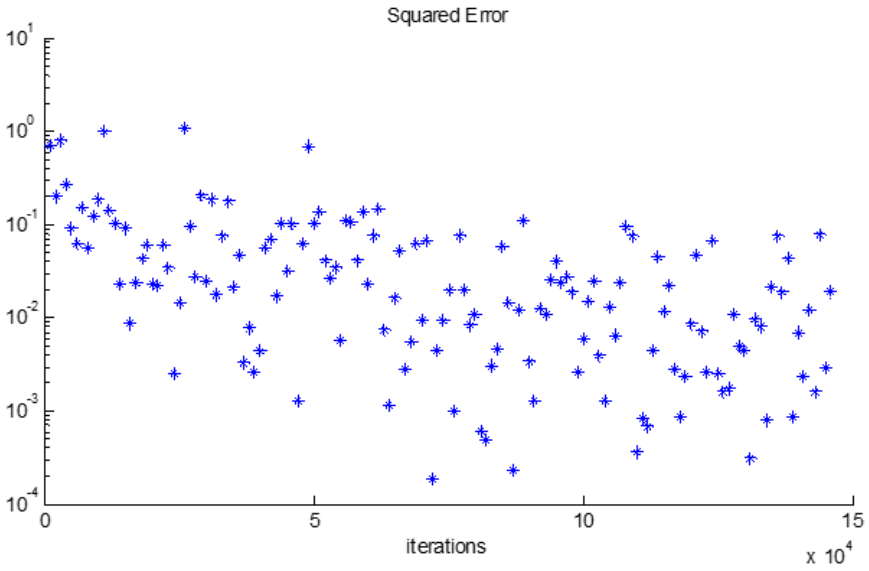


Figure 26: Evolution of the squared error

This kind of relative stabilization on a minimum test score value can be explained by the fact that a limited training base limits the capabilities learned by the neural network. Smaller learning database means less varied examples, less learned situations and so less capability of generalization in the test database. It can be re-interred the calculation with exactly the same parameters except an extended learning database of 6000 examples per class. In fact it consist in taking the entire MNIST training set, so 60 000 pairs of input vectors and their associated output target.

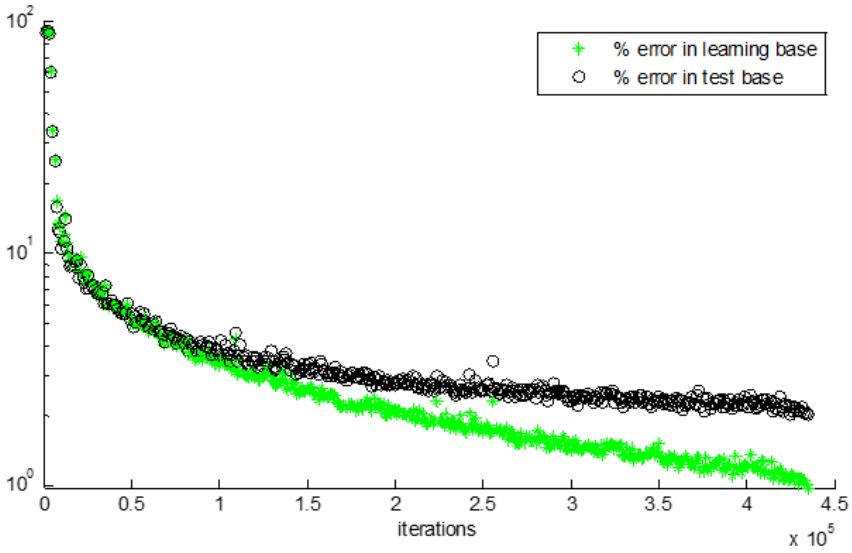


Figure 27: Evolution of classification score on learning database and on test base

With this relatively consequent training database, we can observe that when we reach 99% of good classification on it, the classification error on test database is 98% and keep decreasing. When the training base is quasi learned (99% for time saving), the consequent number and variety of example of the training database base, composed of handwritten digits pictures, written by administrative employees, permit a great capacity of generalization on other database, in our case a test base composed of handwritten digits pictures, written by university students.

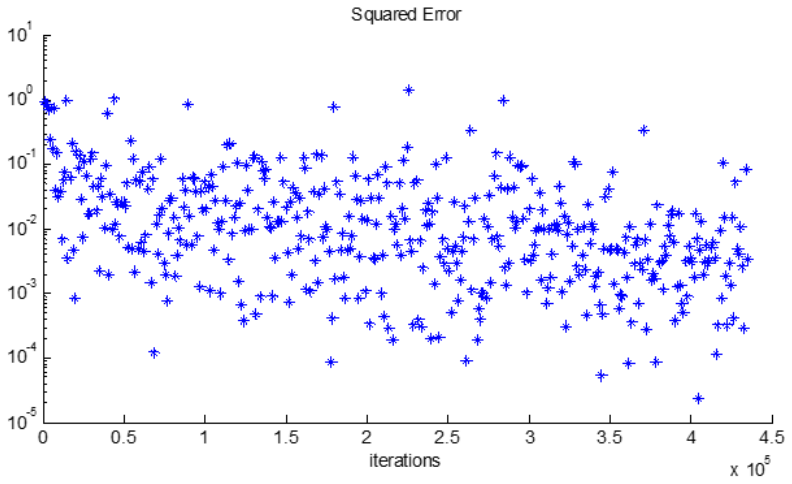


Figure 28: Evolution of the squared error

Well we have tried to fix some parameters of the neural network study, to be able to determine some trends on neural network behavior with respect to its characterizing parameters. For the moment are frozen: the size of the learning database (1000 x 10 examples of input/output) and the number of iterations (100 000).

7.2 Impact of the number of hidden neurons

Continuing with our 0.3 learning rate, we now want to perceive the impact of the number of hidden neurons in our one hidden layer perceptron. To do this, are plotted for different hidden layer architecture the maximum training and learning score obtained for 100 000 iterations. One obvious interest is that we would like the smaller number of neurons to limit the calculation time, and obtain the simplest architecture, synonym of robustness.

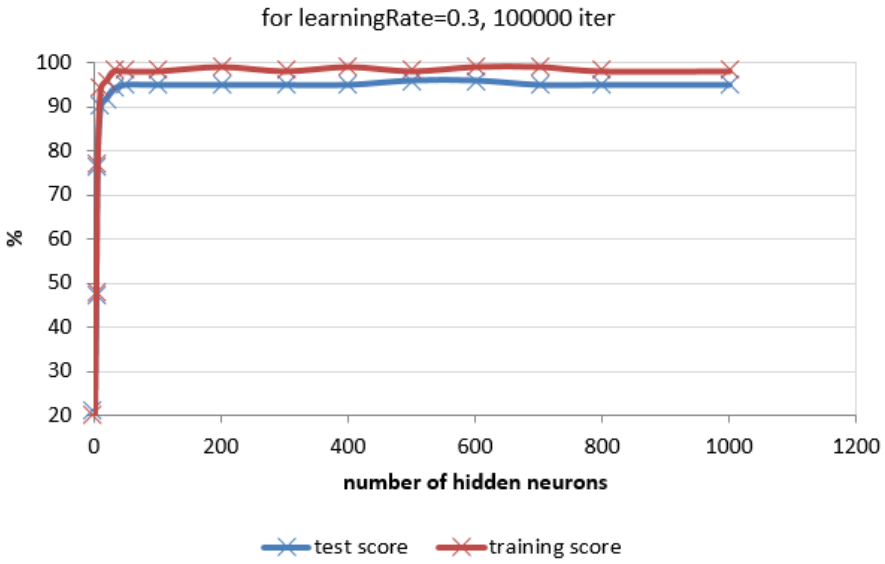


Figure 29: Evolution of the squared error

We can see on 29, page 28, the results obtained. For only one hidden neuron, it's obtained 20% of good classification on the two bases, which correspond to 2 of the 10 digits well classified, after the convergence into the 100 000 iterations. The architecture of the neural network isn't enough expressive. Adding hidden neurons from 1 to 50 permits to highly increase this expressiveness: with 50 hidden neurons we obtain 98% of good classification in learning base and 95% in test base. After that, adding more hidden neurons seems to not have impact on scores. We also can see that training score stay steadily between 98 and 99%, it does not reach 100% of classification score on training database. It's due to the fact we have limited the number of iterations to 10^5, that limits the convergence to learning score between 98% and 99% like it can be seen in 29, page 28.

What if we attempt to resume the current situation with one figure? To compare scores between different databases with different amount of examples, it is needed to impose a common learning advancement, like we have done on figure1. This learning advancement is in fact the learning score in percent that quantify the assimilation of the training database by the neural network. Equalize this examples assimilation is necessary to compare the real impact of the training base constitution. In effect, the assimilation of 97% of a 6000 examples per class training database takes more iterations than to obtain 97% of a 100 examples per class training database. Example of number of iterations to reach 97% of learning for 50 hidden neurons on 30, page 29.

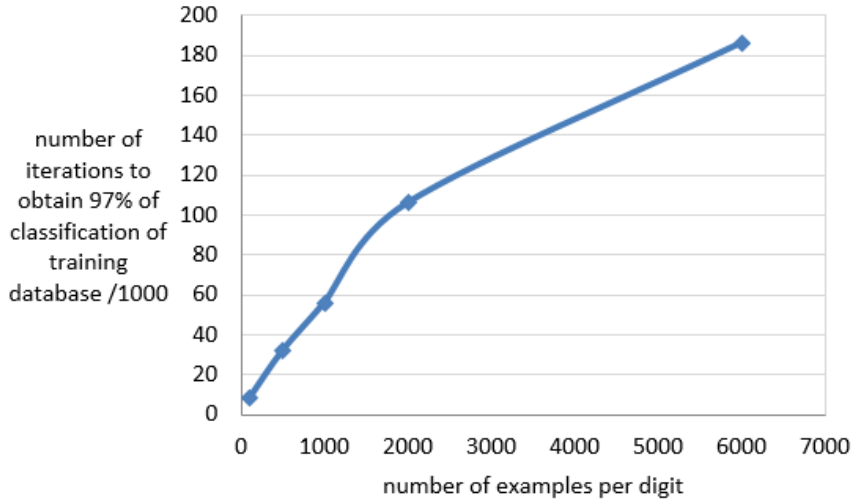


Figure 30: Number of iterations to obtain 97% of learning depending on training base size, for 50 hidden neurons

So we fix the learning advancement to 97% for this quick study to have a compromise between rapidity of learning to this value but also a sufficient training process to obtain honourable test scores. Now lets compare the influence of the database size and the number of hidden neurons on final classification score on 31, page 29:

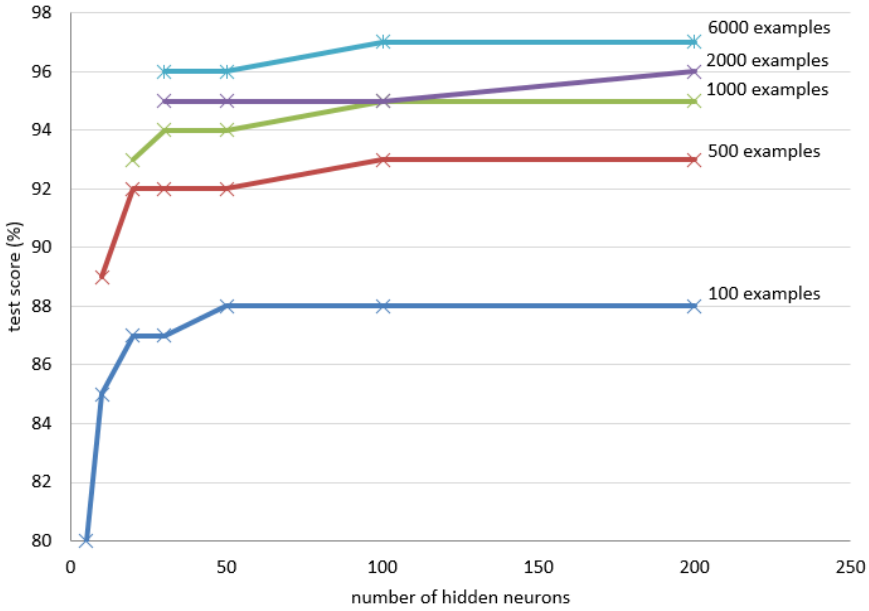


Figure 31: Classification score regarding the number of neurons on the hidden layer and the quantity of examples per digits that compose the training database

We can see various trends: Increasing the number of examples increase the final score in test database. The biggest is the training database, bigger is it variety of examples, and better is the generalization around the test database. The composition of the hidden layer affects the test score.

First of all, if there are too few hidden neurons, the neural network is not able to reach the 97% learning score, and the result cannot appear on our figure. Secondly, it can be seen the increase of test score when we add hidden neurons on a hidden neuron number slot, basically between 5

to 100 hidden neurons. It represents the increase of expressiveness of the neural network. Then, there is a stagnation of the score, independently of the amount of hidden neurons. It's when the top maximum score obtainable for this previous training of 97% on one amount of examples is obtained. With a very diversified training database composed by 6000 examples per class, we can see that we obtain 97% of test classification. It means that theses 6000 x 10 examples contain the specificities that permit to differentiate classes in test database. At the reverse, with 100 x 10 examples training base, it's obtained maximum 88% score with a good number of neurons. This training database induces the neural network has fitted with less variety of examples so the weights and bias will allow less specificities of the test database. Adding hidden neurons increase the expressiveness of the neural network, but if there is a lack of diversity in our training database, the neural network cannot learn to have better generalization performance. In fact, the minimum number of hidden networks has to be chosen depending on the diversity of the training database. We can note that actually the matter is not the number of examples but their diversity, even if in reality the number brings the variety if the base is sufficiently well constituted.

7.3 Impact of the learning rate

For these previous calculations, we have set the learning rate to 0.3 that seems to work well after some attempts for various values. We can attempt to define the learning rate role. With a database of $1000x10$ examples, we fix the number of iterations to 10^5 and the number of hidden neurons to 50. It is obtained the results given in 32, page 30

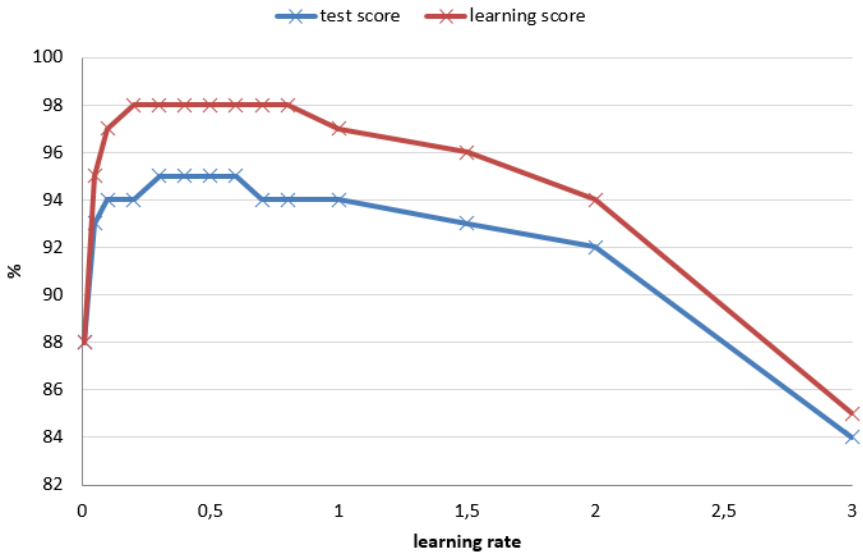


Figure 32: Influence of the learning rate, for 50 hidden neurons, 10^5 iterations and 1000 ex per digit

First of all, for smallest learning rates, their rise from 0.01 to 0.1 induces a consequent increase of both scores on learning and test scores, respectively (88%, 88%) and (97%, 94%). This difference is due to the rise of the velocity of convergence with a bigger learning rate. In effect, for a fixed number of iterations to 100 000, calculations with small learning rate only authorize very limited corrections of the error on each iteration and so there is not enough iterations to consequently change the weights and bias. Actually there is a second situation possible: to stay blocked in a local minimum, due to a too little learning rate. This case is illustrated in the figure 33, page 31.

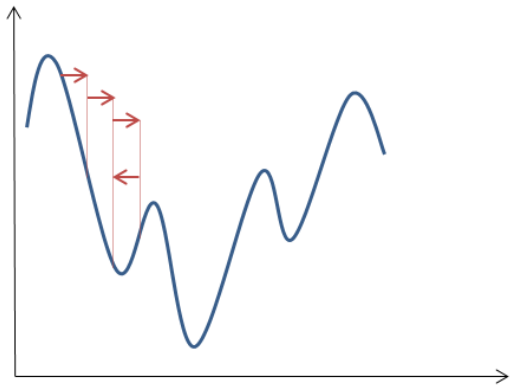


Figure 33: Too small learning rate.

The other tendencies shown on this figure is the fall of the performances with bigger learning rates between 0.8 and 3, respectively (98%, 94%) and (85%, 84%). This can be explained by the fact that because of the learning rate is too large, the optimization takes lot of time to approach the optimum, because it jump around it during a certain amount of time. The second possible consequence of a too large learning rate is that the optimization will not fall into the global minimum. This last case is illustrated in the next figure.

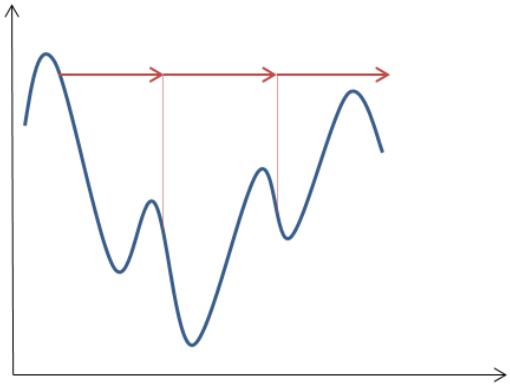


Figure 34: Too big learning rate.

Finally, for a learning rate between 0.2 and 0.8, we can see that the optimum scores is obtained. It is the learning rate range in which the optimization seems approach the most the global minimum.

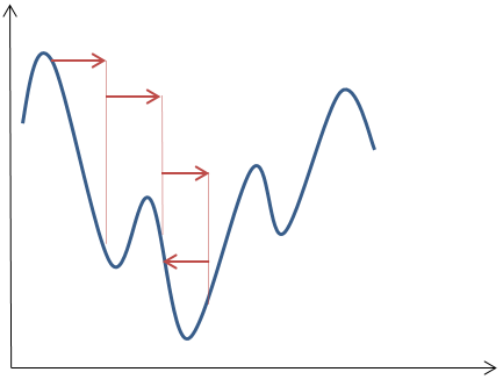


Figure 35: Correct learning rate that permits to fall near the optimum.

The learning rate impact on time and iterations needed to converge is illustrated in the figure. We decide to take an assimilation rate of the learning base of 1000 x 10 examples of 97% to be able to compare times of convergence to a common learning advancement.

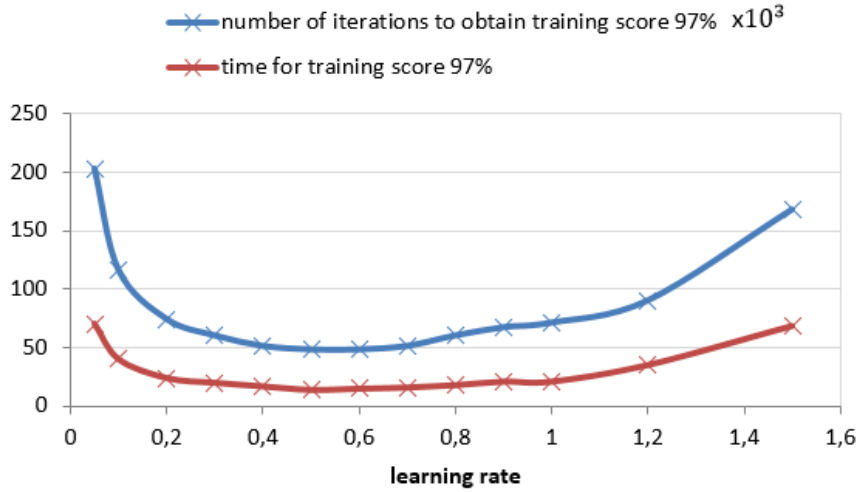


Figure 36: time and number of iterations consideration to obtain the same learning advancement (97%) of the 1000x10 examples training database. With 50 hidden neurons, we obtain for all the learning rates a test score of 94%.

Now we can clearly see that for smallest learning rates from 0.01 to 0.1, the fall of the score is due to the slowness of convergence of the method. For learning rate lower than 0.1, it's needed more than 10^5 iterations to obtain the same learning advancement (97%) than with higher learning rates. For bigger learning rates, between 0.8 to 3, it can be observed two phenomenons : the optimization converge with only a need of more time than for an "optimized" learning rate on the range 0.8 to 1.5, because of the jump around effect of the too large learning rate that induce the error decrease too slowly. Then because of a too large learning rate bigger than 2, the learning advancement to 97% is simply not reachable by the optimization for the reason we have spoken about in 34, page 31.

7.4 The over fitting phenomenon consideration

We do not have had its apparition during this little study but there is a specific phenomenon to take care of in learning methods: The over fitting on training data. It can be induced on our case using a poor training database. Over fitting for 50 hidden neurons, 10 examples for each digit class and a learning rate 0,3 can be seen figure 37, page 33.

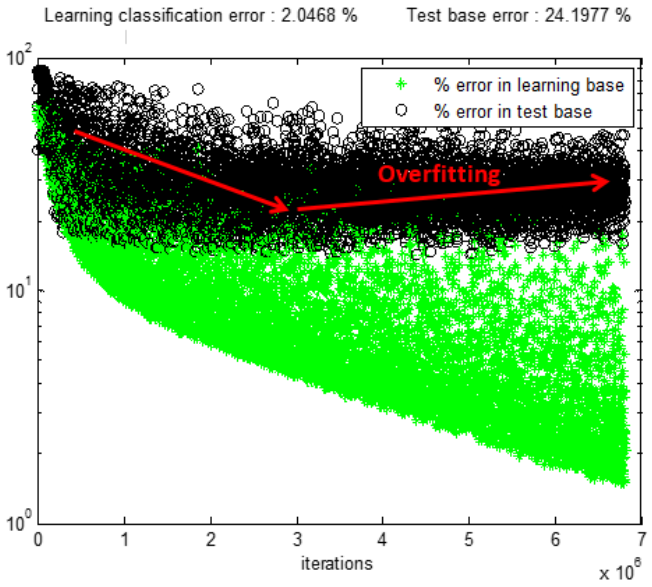


Figure 37: Graphical example of over-fitting on the learning database. The classification error on the test base starts to increase when the classifier fit too precisely the learning examples.

We can see initially the high reduction of the classification error on the test base during the first 5000 iterations, where the minimum is obtained with 26.6% of error on test base. It represents the phase in which the neuron is learning the training database, sufficiently to be able to discriminate the various classes with this score. Then, the stochastic gradient optimization algorithm keep running reducing the squared error on the examples. The training database of 10 x 10 examples is entirely discriminated by the neural network before the 5000th iteration, and then start the phenomenon of over fitting. In our case, we can see that for $4.5\text{x}10^5$ iterations more, the classification error on test database increase a 0.9%.

So it is important to stop the learning phase before than over-fitting occurs, to obtain optimal weights and bias. The over fitting can be reduced using a bigger learning rate that doesn't permit to the neural network to be a "very fitting classifier" like we can see on the figure above, because it limits the fall into smaller minimum for the error, like we have already told in gradient descent principle. The calculation with a learning rate of 0.6 confirm the suppression of the over fitting effect.

8 The results

To test the entire performance of our algorithm, we try with all the training database of 6000x10 examples, a number of iteration elevated, 1000 hidden neurons and an adapted learning rate value (0.3). After $2.5\text{x}10^6$ iterations, we obtain 98,34% of good classification on test base after a learning score of 99,7%. So a test error rate of 1.66%.

Classifier	Preprocessing	Test error rate (%)
2-layer NN, 300 hidden units, mean square error	none	4.7
2-layer NN, 1000 hidden units	none	4.5 (year 1999)
2-layer NN, 1000 hidden units	**none**	**1.66**
2-layer NN, 300 HU	deskewing	1.6
2-layer NN, 800 HU, cross-entropy, elastic distortions	none	0.7
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU)	none	0.35
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23

Table 1: Comparison of the result obtained and benchmark examples on MNIST from [YL98].

Our algorithm implemented through this project doesn't have very good performances compared to the lasts versions of advanced variants using preprocessing, advanced error calculations or convolutional neural networks. In fact, this is not the matter here. The main idea was to understand how this kind of algorithm works on a first application.

Concerning the learning phase, after $5x10^5$ iterations, it's obtained a test score of 97,63% for a learning score of 98,6%. For 11 minutes of learning phase, $1,2.10^5$ iterations passed, the equivalent of two time the entire learning database, the neural network obtains 97% of good classification. The last few % are the longest to obtain. The classification score decrease dramatically on the beginning but then decrease slowly and slowly.

Figure 38: examples of the test base well classified to the left, and examples bad classified on the right with the determined class by the neural network in index.

Concerning the examples classification, it's interesting to look at the good and bad classified digits. Let's see them on the next figure 38, page 34. We can see on the first figure a sample taken from the 9834 well classified examples, and on the second figure a sample of ones of the 166 others.

Except digits very difficult to classify even for an human, the well and bad classified digits are very similar an do no have a very different aspect. It's maybe because the learning database is not sufficiently diversified, it should need more different writings of more persons.

We have said that adding neurons increase the expressiveness of the network. One interesting way to visualize this expressiveness is to visualize its weights after learning phase. For each neuron of the network, we plot the weights influencing signals in the inputs of each considered neuron 39, page 35.

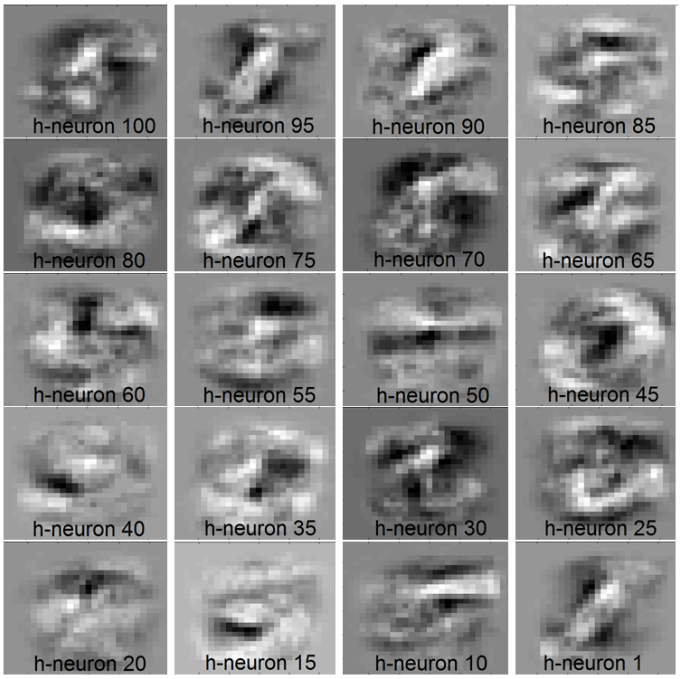


Figure 39: Graphical representation of hidden layer weights after the training session.

It can be see the parametrized neurons, for in black the low weights and in white the higher weights. With this representation, it can be observed for each hidden neuron the pixels of interest, an the zones of interests for the data extraction. After that, the output layer only has to superpose several of these specific zones to be able to reconstruct a digit classification.

Doing the same on the output layer, it can be seen the outputs of the hidden layer which matter to each digit class. For example for the 1 class, the hidden neuron 10 is very representative. One graphical representation is proposed in 40, page 36.

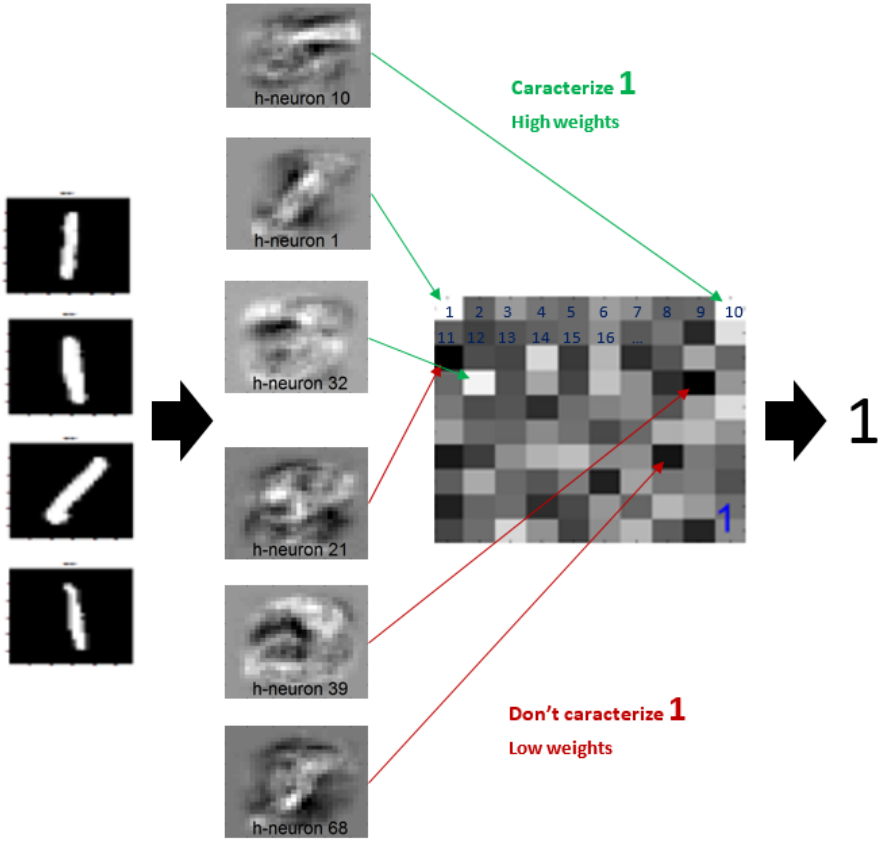


Figure 40: Graphic representation of the Neural network work to recognize the digit class "1"

This possibility to display graphically the "decisions" of the neural network has motivated the treatment of data as picture processing for the next application.

Part II
The autonomous driving application

9 Introduction

9.1 The context

It has been intended to determine the main parameters that induce the cut-in critical situation:

- First of all, the driver(s) behaviour(s). In effect, depending the humor of the driver, or his driving habits, the way he drives is different. If he is stressed or angry, or if he has a sportive way to drive, it may be violent and sharply. That will induce a higher probability of dangerous scenarios, like changes of lanes abrupt and dangerous choice of his course through the traffic. It can be said that the cut-in is more or less planned by the conductor.
- The road configuration is also a factor in the cut-in apparition. Each changes in the infrastructure that induce the insertion on an other lane increase the risk of cut-in occurrence. For example, a vehicle on the insertion lane of a highway can does a cut-in on the right of our vehicle if it speed is higher. The same on a roundabout or a reduction of the number of lanes of the road.
- A third factor that can be proposed is the behaviour of the vehicles in our vehicle surroundings. In effect, the cut-in procedure, or more generally the change of lanes procedure is launched by a conductor because of the situation in which its vehicle evolves is favourable or encourage this action. It can be simply the situation in which the conductor has a higher speed that the vehicle he is passing, and there is sufficient space ahead of the vehicle passed for the insertion of his vehicle. Depending on the free space available, and the evolution of the surrounding, the insertion on the new lane can be done more or less sharply, doing or not a cut-in to the vehicle passed.

Figure 41: The cut-in event is implicated by three main trigger elements.

9.2 The idea

It is searched to be able to anticipate the cut-in situations. There is to define from what kind of information should we prevent this critical situation. In the scope of this project, the available data comes from the sensors that read the surroundings of the ego vehicle. As seen on the introduction part, the fusion of each sensor data permits to reconstruct the environment around the ego vehicle (our vehicle). Additionally, as seen in the "Constraints" section, it is searched to limit the resources. So it was chosen to not take into account the historic of the data. It's taken only the data of an instant t. The motivation of this choice is the fact that an human, on a car, regarding the positions, assessing the speeds and accelerations of each vehicle that surrounds him, can be able to predict

if there will be a change of lane or not. Finally, the temporal evolution of the situation is included in the speeds and accelerations informations.

From the three main cut-in occurrence factors proposed previously in 9.1, page 37, and considering the constraints involved, there is to determine how can be extracted the cut-in occurrence before it occurs.

- From the first point, simply depending on the driver behaviour, prevent the cut-in isn't simple, because it deals with the prediction of the human behaviour, depending on his humor. In our project, we don't take into account the humor of the conductor but the data from sensors detecting the vehicle surroundings. A cut-in manoeuvre initiation in this case can be detected regarding the physical variables from sensors as vehicle position to the lane, lateral speed, lateral acceleration. However it should needs more data historic to be able to predict, taking into account and recognizing from our sensors the behaviour of the driver before the cut-in occurs. This can be done with recurrent methods as recurrent neural networks, that are used for example in word prediction in a phrase, or prediction of the evolution of a variable in the financial domain. In this project, saving an historic time suit of the data evolution doesn't fit with our constraints in resource savings.
- The second point deals with the infrastructure factor. To take into account the infrastructure involvement into the cut-in occurrence probability, it will be needed a way to know the kind of infrastructure on which the vehicle evolves. This information could be obtained from the communication infrastructure-to-vehicle or from a labelled GPS cartography listing for each GPS position the kind of infrastructure. We will neither take into account this infrastructure factor in this present work.
- Here comes the third point. Considering the traffic around our vehicle, the positions, speeds and accelerations of each vehicles should give us the capability to prevent the cut-in relatively early for some situations. One example that could be interesting is the situation in which a low speed vehicle is on the right lane, ahead of a higher speed vehicle. Our ego vehicle is on the left lane, 10 meters behind the low speed vehicle. The high speed vehicle, because of the low speed vehicle, can be called on to pass it by the left, doing a cut-in to the right of our ego vehicle, the high speed vehicle passing behind the ego vehicle and the low speed vehicle. It this situation, if the high speed vehicle doesn't reduce its speed, it's seems easy to prevent in advance the cut-in event, regarding the difference of speed of the low speed and high speed vehicle.

Considering essentially this last point, the main objective of this project is to extract from this kind of situation a preventive estimation of the cut-in from the entire traffic context around the ego vehicle.

We will train our algorithm on classify situations (cut in or not) regarding the position, speeds and accelerations of each vehicles, at the instant t, in a defined window around the ego vehicle.

We want to reproduce the abstract process of the human brain that permits the experimented conductor to assess the possibility of cut-in.

10 The database conception

To be able to train our algorithm on this application, a large amount of data is necessary. As it was seen in part 7.1, page 25, the more data we have, the best will be the training. The first idea was to train on real measured values from prototype recorded database. Unfortunately, the database was not available, so it was decided to generate a simulated database to do a proof of concept. To generate a suitable database for the training process of the algorithm, there is to effectuate various processes seen on the next figure 42.

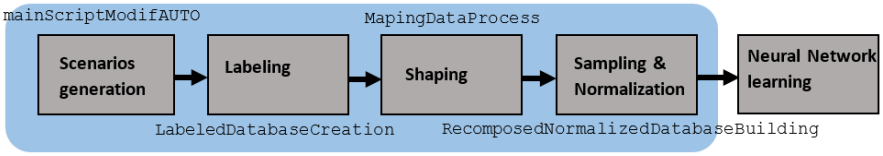


Figure 42: The various steps (and scripts) for the database conception.

10.1 Scenarios definition

A database sufficiently consequent and diversified is the key for learning algorithms. The first think in mind was that we want to predict lane changes, whether in our left or right from the whole data from the context around the vehicle. It means that we want the method to extract characterizations of a cut-in to come up looking at one vehicle, (its lateral acceleration for example)but also we want the learning method to be able to extract interactions between vehicles that warn of a possible cut-in (differences of velocities between two vehicles, their positions, for example). So the first step is to define various cut-in scenarios. We have defined four main cases from daily observations in our roads :

- No cut-in. This is the most common case when we are on the road. There is much more instant t in which we are not overtaken that time t in which a vehicle is overtaken.

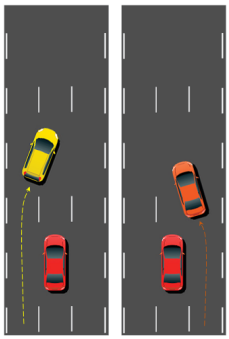


Figure 43: Simple cut-in from the left and from the right.

- Simple cut-in. The vehicle comes from the right or the left lane and comes ahead on the ego vehicle lane, figure 43, page 39. This case recover insertion cases, normal and violent overtaking in both sides. It represents a cut-in case referred to the conductor behaviour or more generally a cut-in initiated independently from the traffic context, excepted the need to have sufficient space ahead of the passed vehicle to insert-in.

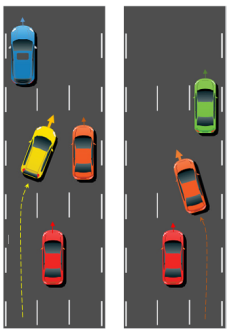


Figure 44: Cut-in induced by a vehicle passing a lower speed vehicle ahead our ego vehicle.

- "Turtle" case. It's a first scenario introducing interactions between vehicles. This case refers to the situation when a vehicle comes with a significant higher speed that the car ahead of

it. The human mind is able to think that there is a high probability that the higher speed vehicle conductor has planned a change lane. Figure 44, page 39.

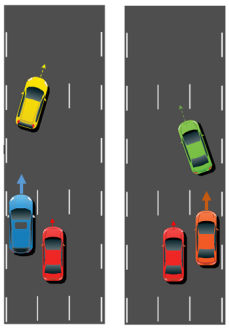


Figure 45: Cut-in occurring when the vehicle is followed and "under pressure" by an other.

- "Glued" case. This last case refers to a possible variant of the precedent case. The lower vehicle driver may start a changing lane move as he can be afraid regarding the incoming vehicle behind itself. 45, page 40.

From these short descriptions, we want to define these scenarios into the simulator.

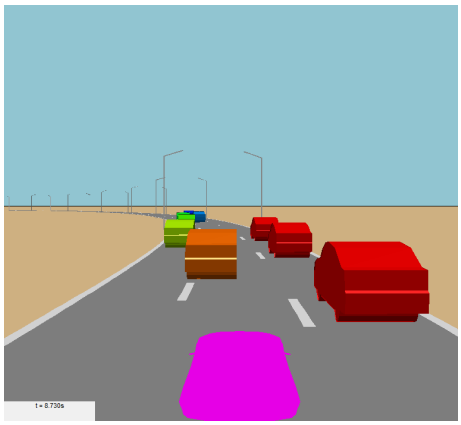


Figure 46: A driver view from the ego vehicle proposed by the simulator.

The simulator available permits to define lot of parameters defining the environment around our ego vehicle. The number of lanes, type of lines, the shape of the road can be fixed. Other vehicles are introduced around our ego vehicle defining their initial position in the simulation, their speeds and lanes at each time. The main objective of this simulator is to generate data from sensors on the ego vehicle 47, page 41. Thanks to this program, we have ready to order sensor outputs to our project.

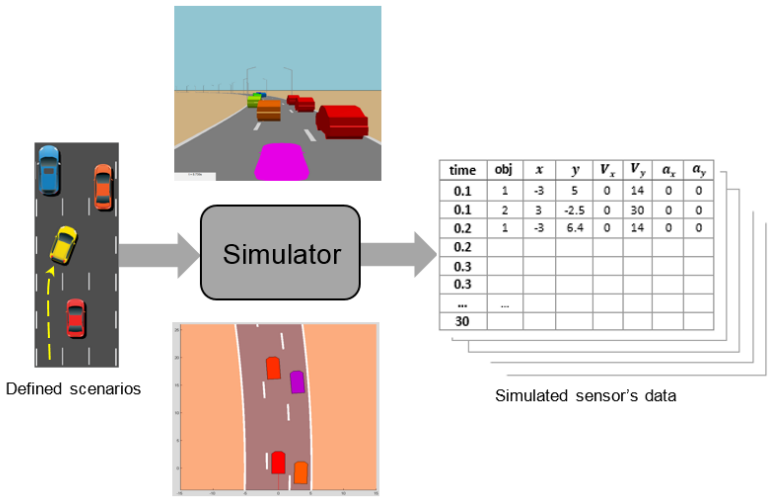


Figure 47: From defined scenarios and some random variations into its parameters, the simulator generates data of a perfect sensor fusion.

To obtain diversity into our examples forming the generated database, additionally to the various possible scenarios seen before, each parameter of interest is determined and vary randomly at each example. An overview of theses parameters is proposed in 48, page 41.

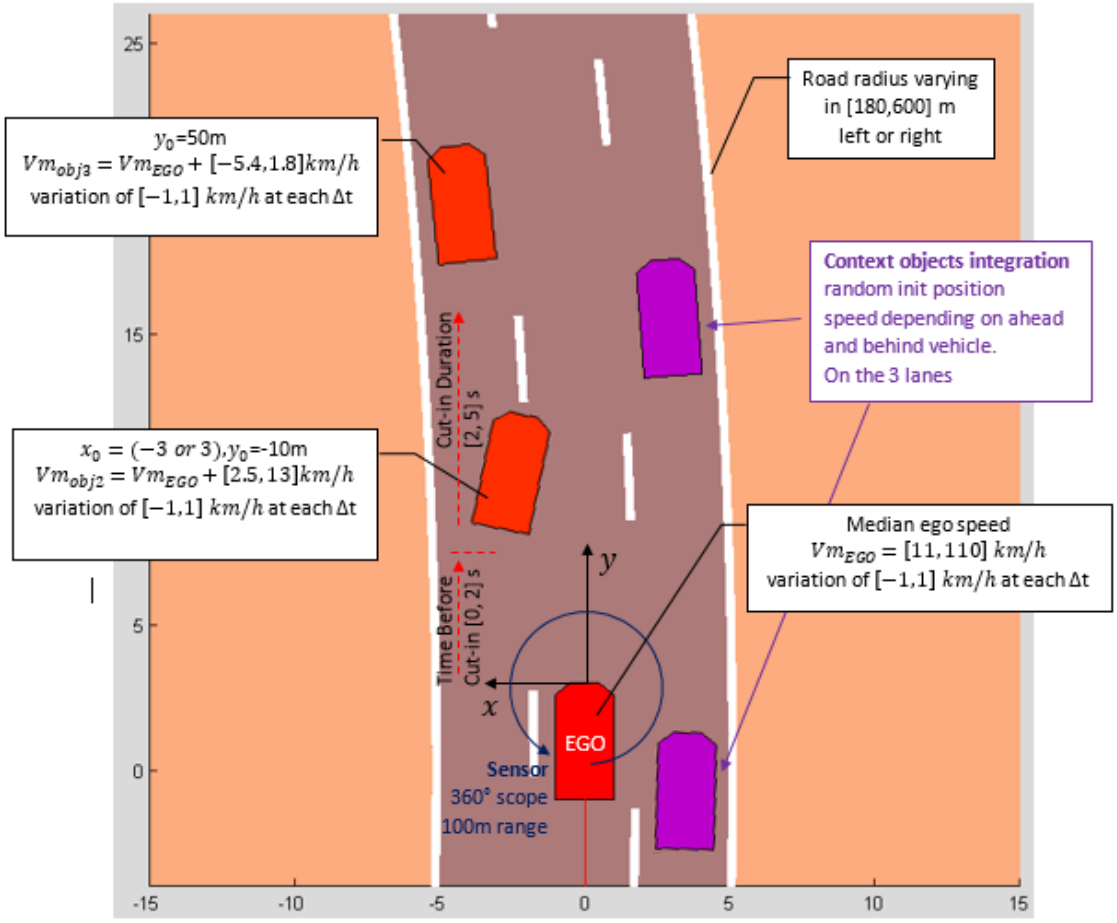


Figure 48: Overview of varying parameters in scenario generation. An additional wobbling function permits to generate wobbling trajectories for each vehicles, permitting more diversified, realistic and robust data generation.

The road is defined as a constant curve ether to the left or to the right. Its radius is defined randomly to obtain variability. If the road stays unchanged for each scenario (for example a straight line), the interest of the neural network algorithm is lost : in effect the simple detection of a vehicle position between the two lanes (independently the speed or acceleration) should be sufficient to define the cut-in. In the same way, the interest of curves is to induce lateral speeds ans lateral accelerations. Without curves, the simple lateral speed or acceleration reaching a threshold would be sufficient to define the cut-in occurrence. The interest of the neural network in this application is precisely to work on extended and complex situations, where the rules based algorithms are limited. In addition, wobbling trajectories and speed variations are generated for each vehicle to make the work harder for the classifier, that will need to extract from inputs data the deep specifications that define the cut-in scenario.

10.2 The labelling

The stage of labelling simply consists in determining the target vector that determines the class in which each input vector belongs. In our application, it consists in for each time t of the scenario, calculates the overlap of the changing of lane vehicle, and associates one of the 3 classes (no cut-in, cut-in left or cut-in right) referring to this.

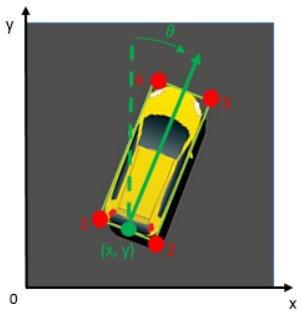


Figure 49: Overview of varying parameters in scenario generation. An additional wobbling function permits to generate wobbling trajectories for each vehicles, permitting more diversified, realistic and robust data generation.

Knowing the road lines position in the simulated scenarios, the idea was to automatize the labeling task. First, I developed a function that gives us the four corner coordinates from the coordinates of the center point of the rear vehicle axle and the yaw of the vehicle. The 49, page 42 permits to visualize the problem. Secondly, I used an overlap calculation function developed by the Renault

data fusion team. This function takes in input the radius of the road and the position of the four corners of the vehicle. Remember we defined the road as a constant radius circular one. It gives us in output if it is a right, left cut-in or no cut-in situation, and the overlap associated (% of object area into the ego vehicle lane).

Concretely, the labelling step consists in concatenating the proper target vector to the input vector as we can see in 50, page 43. From this manipulation, it's obtained the pair of input/output necessary for the neural network learning phase. Is considered a cut-in situation in out case when the overlap is over 25%.

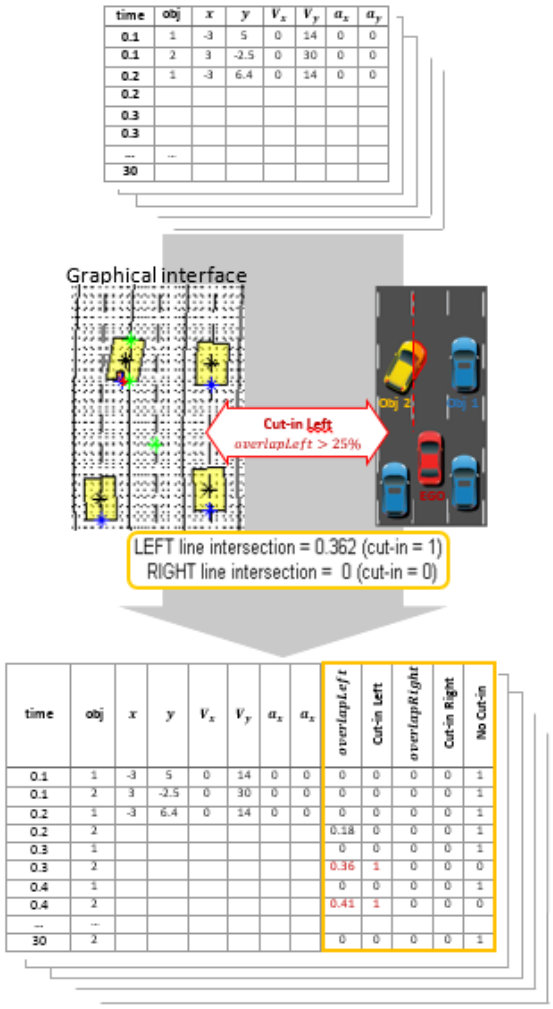


Figure 50: The labelling process consists in adding the classification information for each example.

The finality of the project is to predict cut-in. To classify the examples with predictive capacity, the labelling is done similarly but when a cut-in is averred, the labelling is extended back in time before the time it occurs. On that way, the neural network would find the characterization of this extended cut-in class that contains the averred cut-in ($overlap >= 25\%$) and examples that represent the instants before the cut-in. Of course there would be a limitation on the prediction of the cut-in referring on the existence of differences between a no cut-in example and a cut-in example. In a violent cut-in case, the prediction is limited because there is no difference between a no cut-in case an a cut-in case parameters till the last time when the cut-in manoeuvre starts.

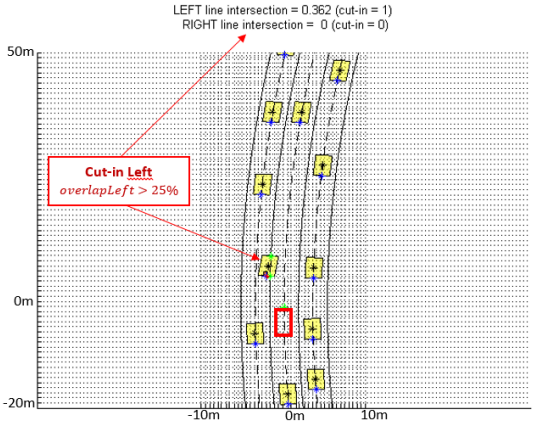


Figure 51: A view of the labelling display developed that permits to verify the proper functioning of the labelling function.

The creation of a graphical display figure 51, page 44 permits to verify the correct classification of the scenarios, and correct line intersections estimation with the associated values of overlap.

10.3 The shaping in map form

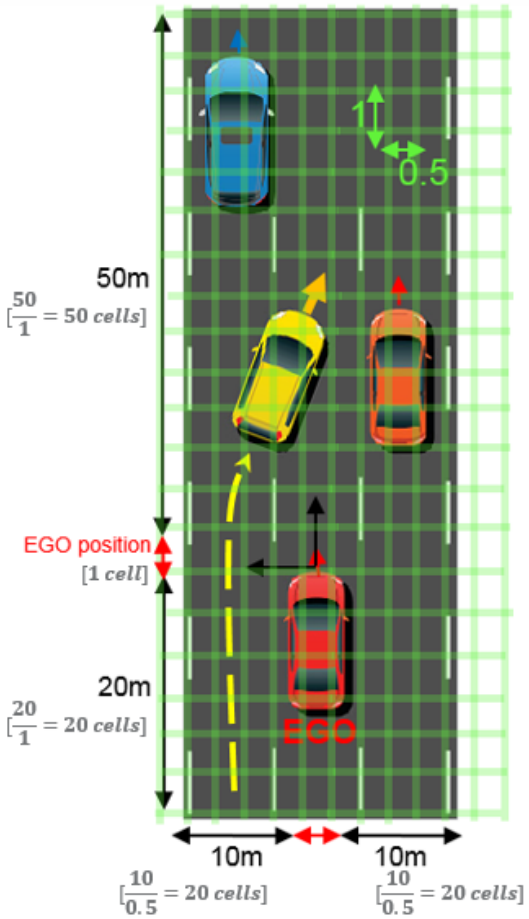


Figure 52: A window around the ego vehicle is defined as it can be seen on this figure. The definition of the unitary cell dimensions is fixed considering the necessity of higher precision on lateral dynamics for cut-in prediction. It's obtained a 71x41=2911 cells map.

The original idea of our project is to treat data as maps. This direction was taken after a thought on how to build the input and output vectors. In effect, the neural network needs a fixed number of inputs and outputs. A way to fix the number of elements of the input vector is to convert ours data vectors in matrix shape representing the 2D surroundings. On that way, whatever the number of vehicles presents in the window of interest, the number of inputs stay unchanged. Further, an advantage of this choice is the possibility to look at the tuned weights as 2D maps pictures as we have done in the handwritten digits recognition application. It gives us the possibility to visualize the extracted features.

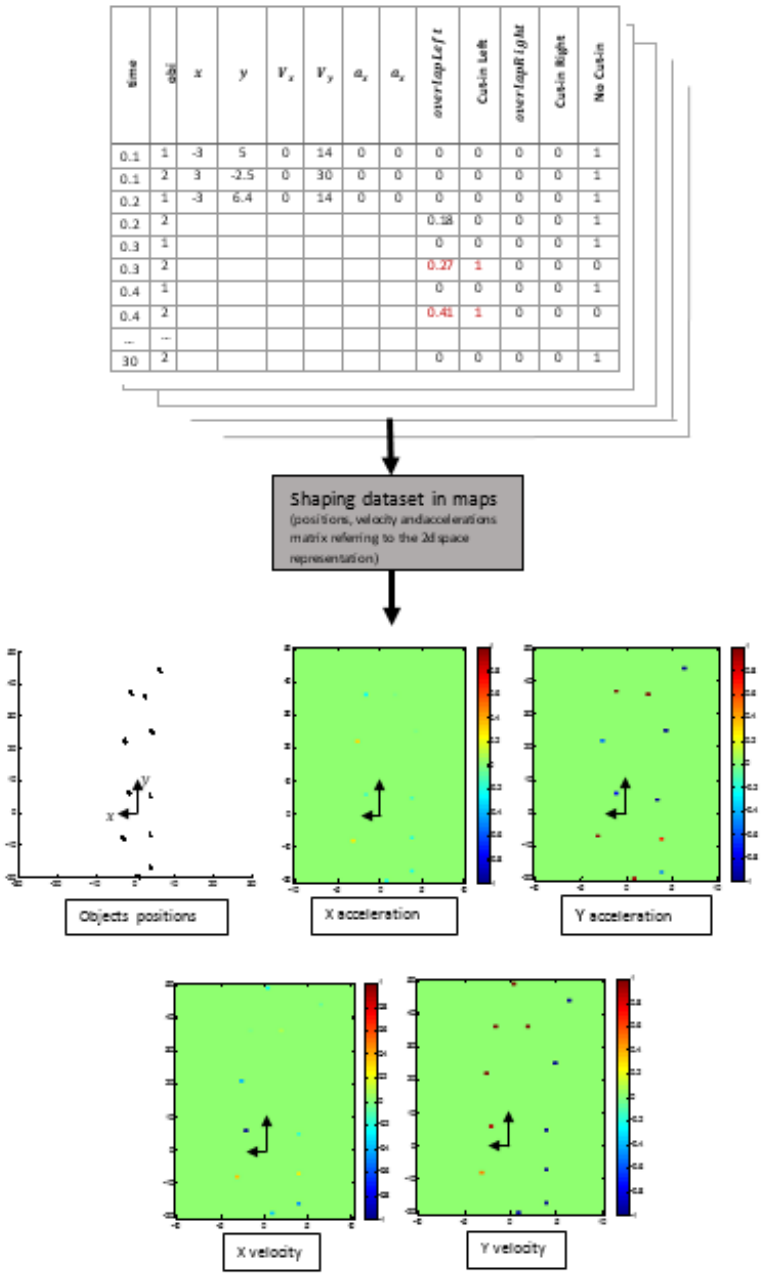


Figure 53: The mapping process consists in shaping data into matrix representing the ego 2D vehicle surroundings. The positions map on this figure has a lateral scale in [-30,30] meters, whereas the others have a lateral scale in [-10,10] meters that explains the lateral deformation of the positions regarding the others maps.

It was chosen to take in input the position information, the lateral and longitudinal speed, the lateral and longitudinal acceleration. From these data, the neural network has to be able to extract

features that discriminate the various classes (left cut-in, right cut-in, no cut-in). Regarding the five maps built, it can be thought that the position information seems to be in each map, and that the position map isn't useful because of the redundancy of this position information (each point on maps is a vehicle). However, if the position map isn't taken in input, in the case of a vehicle with similar speed and acceleration than the ego vehicle, the ego vehicle couldn't be able to see this vehicle.

The speed and acceleration consideration permits to take into account the temporal evolution of the vehicles, and their consideration on both longitudinal and lateral axis permits to integrate the rotational information (yaw). At the end of the map shaping process are obtained an occupation map built from the list of the (x,y) positions of each vehicles for each instant t. Concerning the speeds and accelerations, each cell of the map reflects the amplitude of the concerned variable of the vehicle occupying the cell. It can be seen for example that on the Y velocity map (longitudinal velocity) figure53, the vehicles on the right lane have a lower speed than vehicles on the left lane, whereas the distribution of lateral accelerations X and longitudinal accelerations Y values are defined by the road curve but also by the wobbling trajectory.

10.4 The sampling

This data process was added after have encountered the accuracy paradox problem. Launching the learning phase after have generated a simulated database, it was obtained directly a very high score (90% of good classification) on the very firsts iterations of the algorithm, but then the error never decreased. The problem here was that the database contained unbalanced number of examples for the three different classes (left cut-in, right cut-in, no cut-in). Unbalanced classes database induces the algorithm learns only weights and bias that permits to gives the output vector value desired for the more represented class. In our case, the no cut-in class was over represented (more than 90% of the examples). The algorithm learned to provide an output vector close to the target output value representing the no cut-in classification, and that independently of the input nature. The solution implemented to counter this problem was the under-sampling of the database.

The idea is to modify the database to balance the number of no cut-in, right cut-in and left cut-in examples [Bro15]. In our case it's simply determining the number of left cut-in examples and right cut-in that are very similar between these two classes, and reducing the number of no cut-in examples to obtain a balanced classes database.

10.5 The normalization

The objective is to normalize data in $[-1, 1]$ referring to the *tanh* activation function and the kind of data to treat. In practice, the choice was done to normalize first in $[0, 1]$ and then adapt in $[-1, 1]$. To do the $[0, 1]$ normalization, there is the necessity, first of all, to take into account the entire database.

Concretely, we want the minimum and the maximum of each variable of interest on the whole database of examples (X speed, Y speed, X acceleration, Y acceleration). Something to take into account: the position information is defined by a Boolean occupation grid with values 0 or 1, so the position matrix do not need normalization. After the normalization of each variable on the entire database, we concatenate each map of 2911 cells on specific vectors that are concatenated in one input vector of $2911 * 5 = 14555$ elements for each instant t. Then the data is adapted from $[0, 1]$ to $[-1, 1]$ interval.

11 The algorithm adaptation

At each different application some specific consideration have to be done. In effect, depending of the kind of data we need to extract information, which kind of information, there is to reconsider some points of the method. With this second application, it is shown the adaptation of the algorithm from the handwritten digits recognition to the prediction of cut-in. One of theses adaptations concerns the nature of the activation function defining the threshold in each neuron.

In the handwritten digits recognition previous application, the inputs were Grey scales vectors normalized in $[0, 1]$. This normalization was convenient because the information was contained in the intensity of white and black in the pictures. However, this road scenarios classification do no longer takes in input a simple amplitude of white in a black picture. In effect, the main change is that it's needed to extract information as much on the negative speeds and accelerations that in the positive speeds and accelerations. In effect, the neural network algorithm principle is to strengthen weights between high amplitude input and high amplitude output through the entire network. If we keep on with a $[0, 1]$ normalization, for a negative ($V_{MIN} = 0$) to positive ($V_{MAX} = 1$) scale of speeds, the high amplitudes of negatives speeds aren't been taken in account, subdued by the non strengthening of the connexions to low amplitude inputs of the neural network. It can be commented that a zero value input cannot transmit information because it inhibits the weights connected to it.

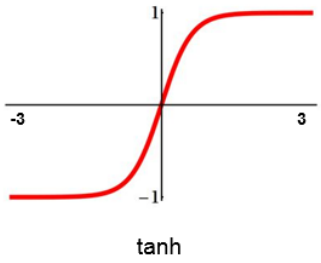


Figure 54: An hyperbolic tangent function representation. It takes inputs in $[-\infty, \infty]$ and gives input in $[-1, 1]$.

So it's chosen to use an hyperbolic tangent (figure 54, page 47) as activation function instead of the sigmoid function. These two functions look the same but the hyperbolic tangent gives outputs in [-1,1]. Thanks to this, the amplitude's information is conserved for negative and positive speeds and accelerations. On this way, modifying the activation function induces an adaptation of the initialization of the weights and bias, but also a modification of the learning rate to compensate a change in the activation function derivative values and the different nature of the data.

The initial weights $w_{ki}(0), w_{pk}(0)$ and bias $b_i(0), b_k(0)$ must be chosen differently on this application because we have changed the activation function. We refer to the paper [XG10] which proposes for an hyperbolic tangent activation function, values in the following interval:

$$\left[-\sqrt{\frac{6}{nb_{neuronsIN} + nb_{neuronsOUT}}}, \sqrt{\frac{6}{nb_{neuronsIN} + nb_{neuronsOUT}}}\right]$$

Concerning the number of hidden neurons for this application, regarding the dimensions involved (14555 inputs and 3 outputs), it was chosen a number of hidden neurons equal to 2. This reduced number of neurons permits, trough cross validation, to avoid over fitting and to reach good classification performance on reduced database. Two hidden neurons is the minimum possible to be able to discriminate three classes. The main preoccupation here is to limit to the minimum the number of neurons for time calculation mater. In effect, 2 hidden neurons means already $14555 * 2 + 2 * 3 = 29116$ parameters to tune.

12 The results

The objective is to predict the change of trajectory: so, as seen page 43, the labelling is done extended one second back in time on a first trial, and extended two seconds back on a second trial. The algorithm should predict with one second of prediction time maximum in the first trial, if the learning phase has permitted the extraction of specific characterizations that define the classes.

12.1 The classes characterizations extraction and classification score

We want to present graphically the classes specifications extraction defined during the learning phase that permits the classification. This observation permits to determine more finely the variables of interest for the application. On the next figure 55, page 49, we can visualise hidden weights (links from the 14555 inputs to the hidden neuron 1 and 2) and the outputs weights (links between the two hidden neurons and the three outputs). We also want to assess the tendency of the weights modification depending of scenarios type. In effect, we learn the weights on a consequent database build with relatively rapid and violent changes of lane figure 55 and 57, and we have compared with a database formed of relatively smoother changes of lanes figure 56 and 58. The second parameter we want to assess the impact is the labelling time prediction. On the figure 55 and 56 following, the database on which the weights are learned is a one second anticipation database.

The weights between the 14555 inputs and the two hidden neurons can be represented as two 14555 cells maps representing the weighting of each spatial information for the data extraction. Between the hidden lane and the output lane, there is six weights : two weights for each one of the three outputs.

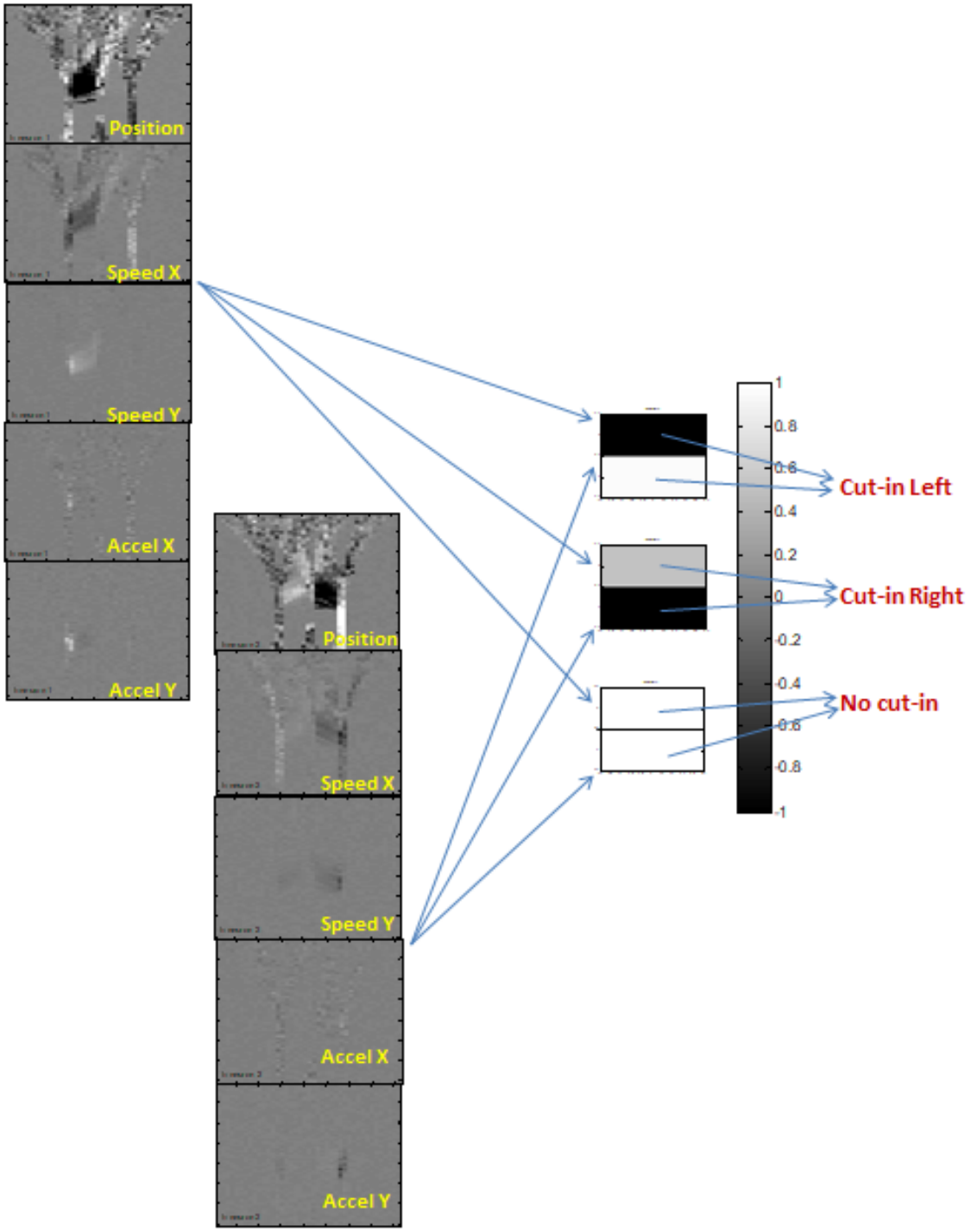


Figure 55: A graphical representation of the learned weights after the training phase on a 1 second prediction database, build with relatively rapid changes of lane.

The first difference we can observe between the rapid case figure 55 and smooth case figure 56 is referring to the longitudinal vehicle acceleration (on Y axis). In the rapid changes case, this variable is taken into account for the classification process, as we can see isolated spots of weight amplitude variation, in the fifths hidden weights matrix part representing the Y acceleration amplitude extraction.

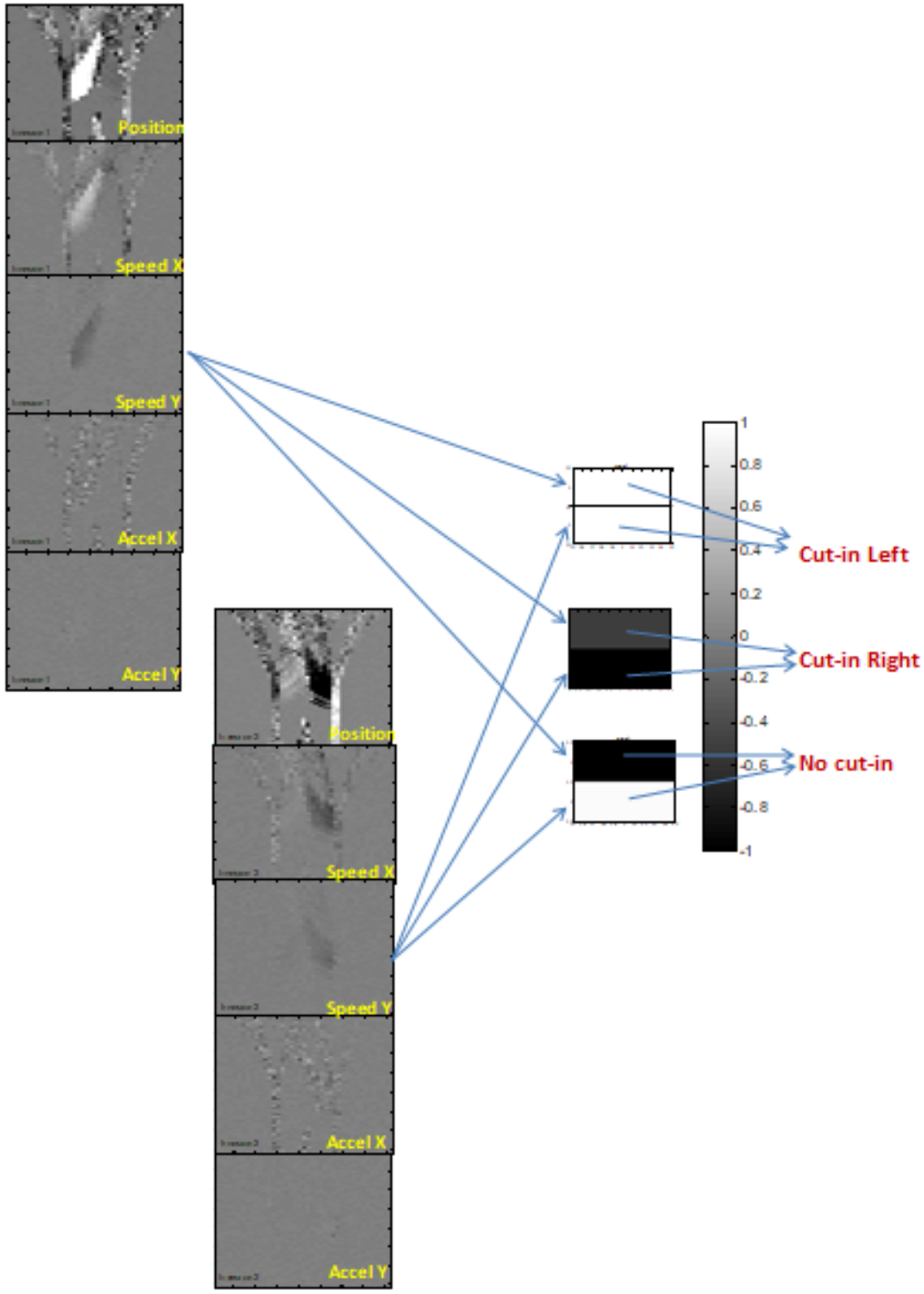


Figure 56: Learned weights after the training phase on a 1 second prediction database, build with relatively smooth changes of lane.

This observation can be done the same way on the 2 second anticipation cases figures 57 and 58. The explanation is that because of the relative rapidity of change of lane, the longitudinal speed of the vehicle is transferred on the lateral axis, that induce the 'violent' fall of the longitudinal speed and so a consequent variation of longitudinal acceleration. On smoother cases, there is no sufficient variation to take into account the longitudinal acceleration as classifier parameter.

The second observation is that there is no real interest in looking at the lateral acceleration (on axis X). In effect, there is no spots shaped, no data extracted for scenario classification. This can be explained by the introduction of the road curvature, but also the wobbling trajectories of each vehicles that induces lateral accelerations. The algorithm cannot discriminate a cut-in situation regarding this parameter that has same amplitude with a no cut-in scenario.

The most influential variables are the position and the lateral speed (axis X) of the vehicles. These two variables express the most the scenario classification, while the longitudinal speed (axis Y) adds a trajectory information.

On the next figures 57 and 58, the cut-in anticipation time for labelling on the learned database is two seconds.

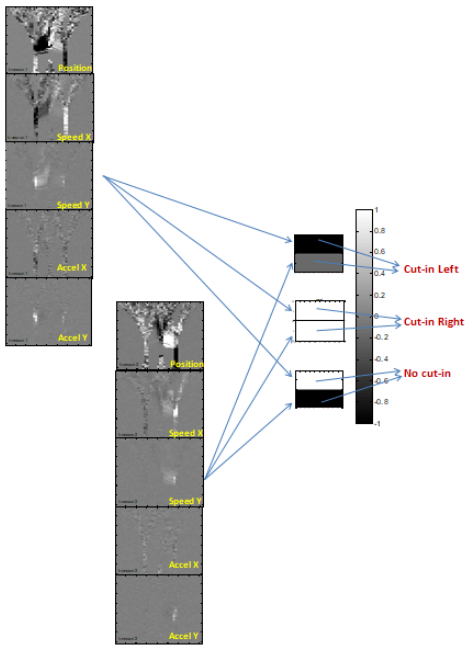


Figure 57: Learned weights after the training phase on a 2 second prediction database, build with relatively rapid changes of lane.

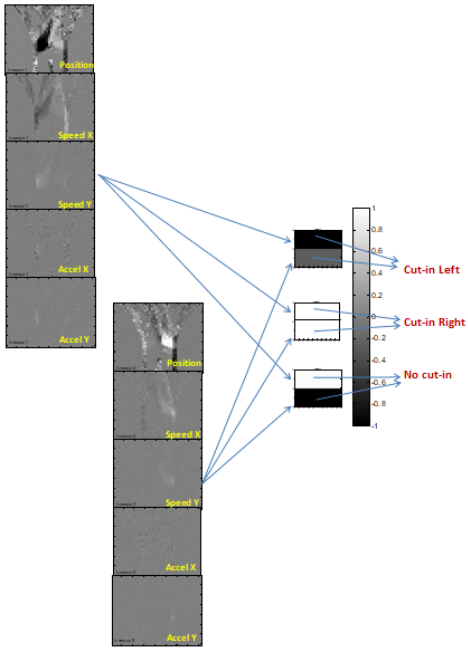


Figure 58: Learned weights after the training phase on a 2 second prediction database, build with relatively smooth changes of lane.

We cannot see lot of difference between the one second prediction weights tuning and the two second prediction weights tuning.

Conclusions: The lateral acceleration X does not provide information to predict the trajectory change (the lateral acceleration takes a similar amplitude in a trajectory change or a curvature of the road). Wobbling trajectories blur the definition of classification by lateral acceleration, giving a more robust classification to the possible hesitations of the driver.

The longitudinal acceleration Y of the vehicle making the trajectory cut is taken into account during rapid changes, where this rapidly decreases by the lateral displacement of the vehicle. This information is, however, reduced by introducing vehicle speed variations. The addition in our simulation of vehicle speed variation add robustness to the classification according to this criterion. The position and lateral velocity X permit to take into account the entire layout of the vehicles making up the surrounding traffic. The variation in the longitudinal speed of the vehicle performing

the cut-in makes possible to define the presence of a trajectory cut.

From these network weights tuning are obtained the following classification scores seen in figure 59:

	RAPID cut-in		SMOOTH cut-in	
	Error :	Time of calculation :	Error :	Time of calculation :
1s prediction	10 %	11h47min	5 %	21h06min
2s prediction	10 %	38h07min	10 %	23h01min

Figure 59: Classification errors obtained on the 1s and 2s prediction databases, and the calculation time necessary to learn.

On the two next figures 60 and 61 can be seen the plots representing the optimization of the weights during the training phase of the algorithm on the 1 second prediction smooth trajectory changes database. This database contains the equivalent of 300 trajectory changes scenarios, each one decomposed in 300 'pictures' of the traffic (one at each 0.1s during the 30s scenario duration), that gives 90000 pictures, and then under-sampled as seen in 10.4, page 46. We obtain a balanced database ready to be learned composed by 14270 pictures : We take 90% (12851 examples) to build the learning base and 10% (1419 examples) for test base.

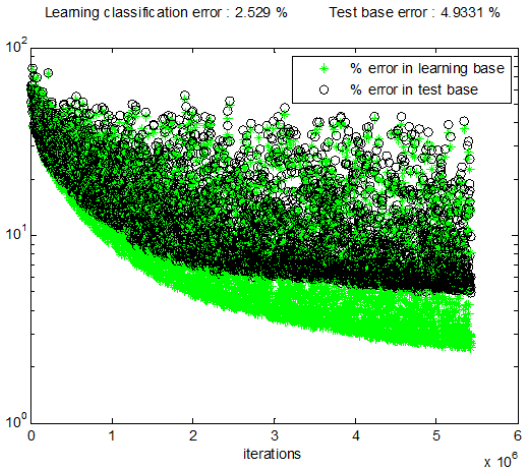


Figure 60: Reduction of the error score during the learning phase. We can see in green the error score on the learning database and on black the test database that permits to validate the algorithm.

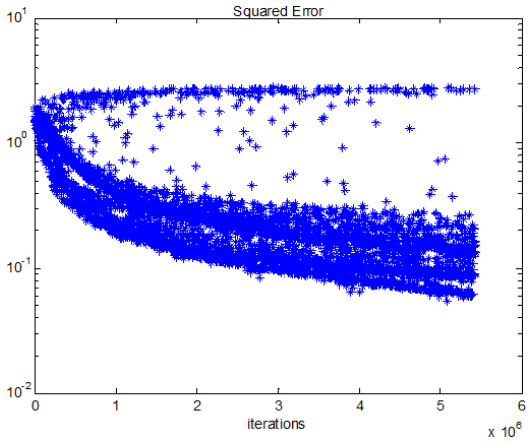


Figure 61: Evolution of the squared error during the learning process on the 1 second prediction database composed of smoother trajectory changes.

12.2 The cut-in prediction results

Regarding the scores obtained in the table 59 page 52, it seems that our algorithm is able to classify each 'picture of the traffic' with a certain accuracy (90 and 95%), including the predictively labelled pictures of cut-in with an honourable advance of 1 or 2 seconds. We want now to verify how the algorithm behaves during an entire scenario of change of lane, taking into account the continuity of the time between each pictures of the traffic at each $0,1$ seconds time-step. To do that we load the weights learned and present a scenario :

We load the weights learned on the 1 second prediction database, and we present to the neural network at each 0,1 second the picture associated to the traffic. We can see on this first scenario figure 62 that the algorithm predict the cut-in at the time $10.2s$ while the cut-in occurs at time $11.1s$.

So the method permits to predict the cut-in 0.9s before it occurs. Time 34.5s we can see that the ground truth define the vehicle in *cut-inRight*, one time step after in *Nocut-in* class and then return in *cut-inRight*. This is induced by the vehicle wobbling trajectory, the vehicle goes out the ground truth zone defining the cut-in case, and return biting on during a lapse of time.

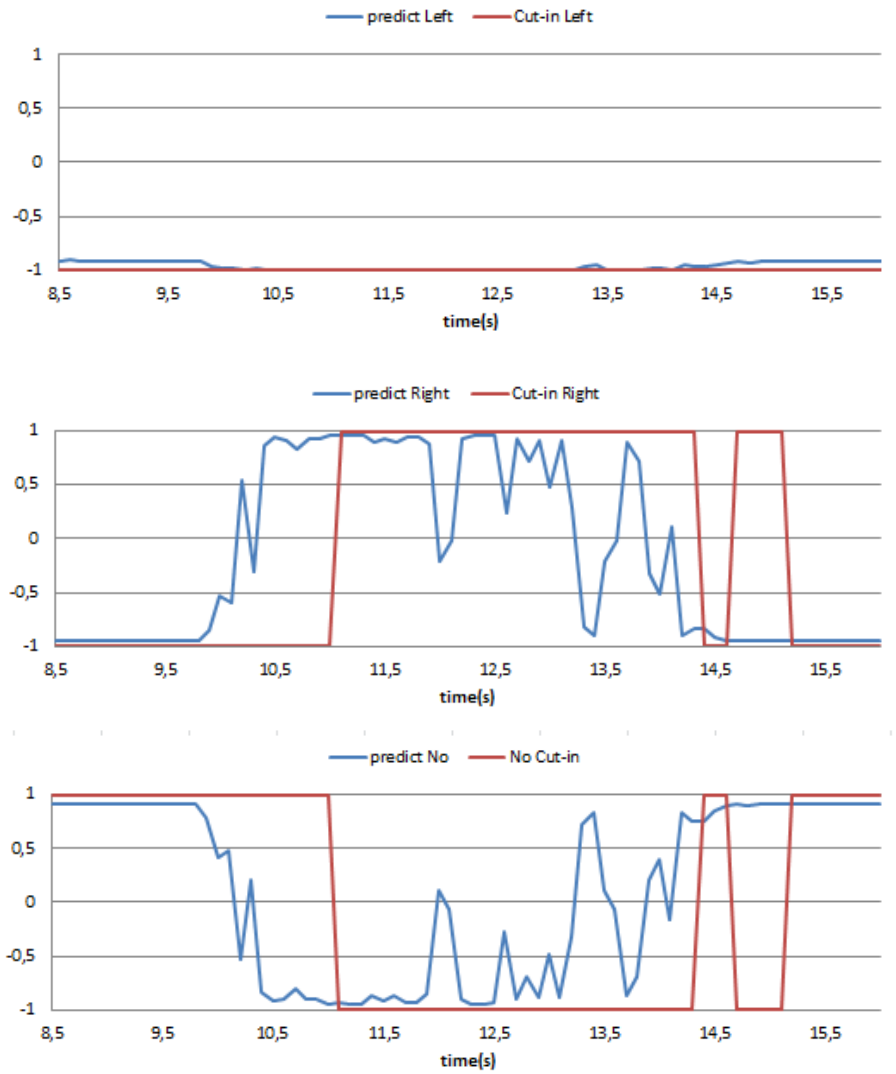


Figure 62: Prediction of the right cut-in on a simple cut-in case, with the weights trained to predict 1s in advance. The algorithm seems to have difficulties to classify on the end of the cut-in procedure (smoother dynamics, position near the center of the lane...).

Now we would like to test one of the great interest of our method : be able to predict cut-in observing the context around our ego vehicle. On this application, the prediction can be done observing the vehicles around and their interactions : a high speed vehicle should pass a lower speed vehicle, whether it is by the right or the left, if it doesn't change its pace. This introduces changes of lane, and the algorithm can extract the fact that a higher speed vehicle is approaching another vehicle as a cut-in feature.

Lets try a scenario introducing the interaction between two vehicle : a 'turtle' scenario figure 62:

We see on that case figure 63 that the prediction is more precise, with a frank step at time 5.8s, and a stable classification through the time to the end of the passing procedure time 11.1s. The ground truth defines the beginning of the cut-in at time 6.8s, so we obtain one second of prediction.

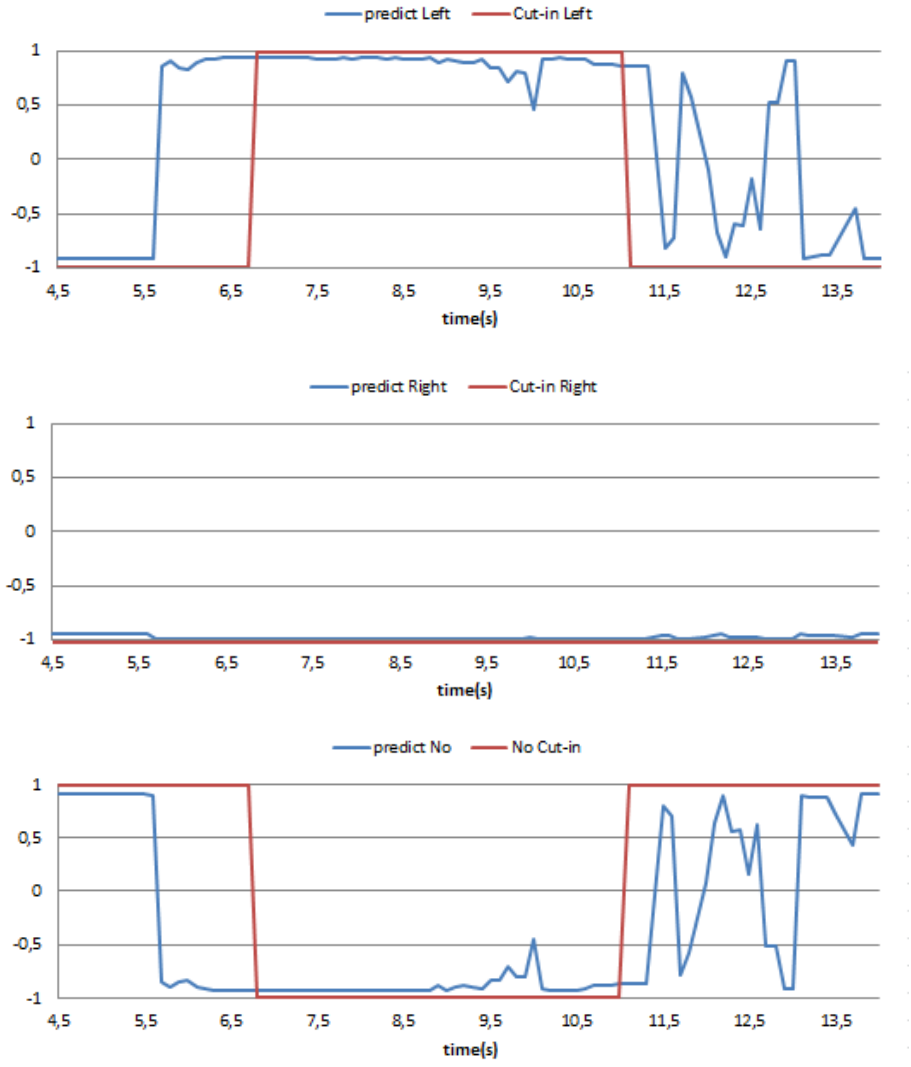


Figure 63: Prediction of the left cut-in on a 'turtle' scenario.

The explanation why this case work very well is surely because the method detects that the 'cut-in' vehicle is approaching a 'turtle' vehicle, and is able to predict relatively in advance that the change of trajectory is necessary to avoid the crash.
We can see again the difficulty to discriminate the classes at the end of the change of trajectory after time 11 seconds. These instabilities are not a problem because it occurs after the cut-in is already detected.

Seen these good results, we want to try a more ambitious prediction. We built a database labelled in 2 seconds prediction, trained the algorithm and tried:

First thing we can see on figure 64: the instability in the classification. In effect, we can observe variations of high amplitude traducing the instability in the classification of pictures which follow each other temporally. Elements that belongs to *Nocut-in* class are classed as *cut-inLeft*, after the event detection but also and mostly before the detection, that is more problematic.

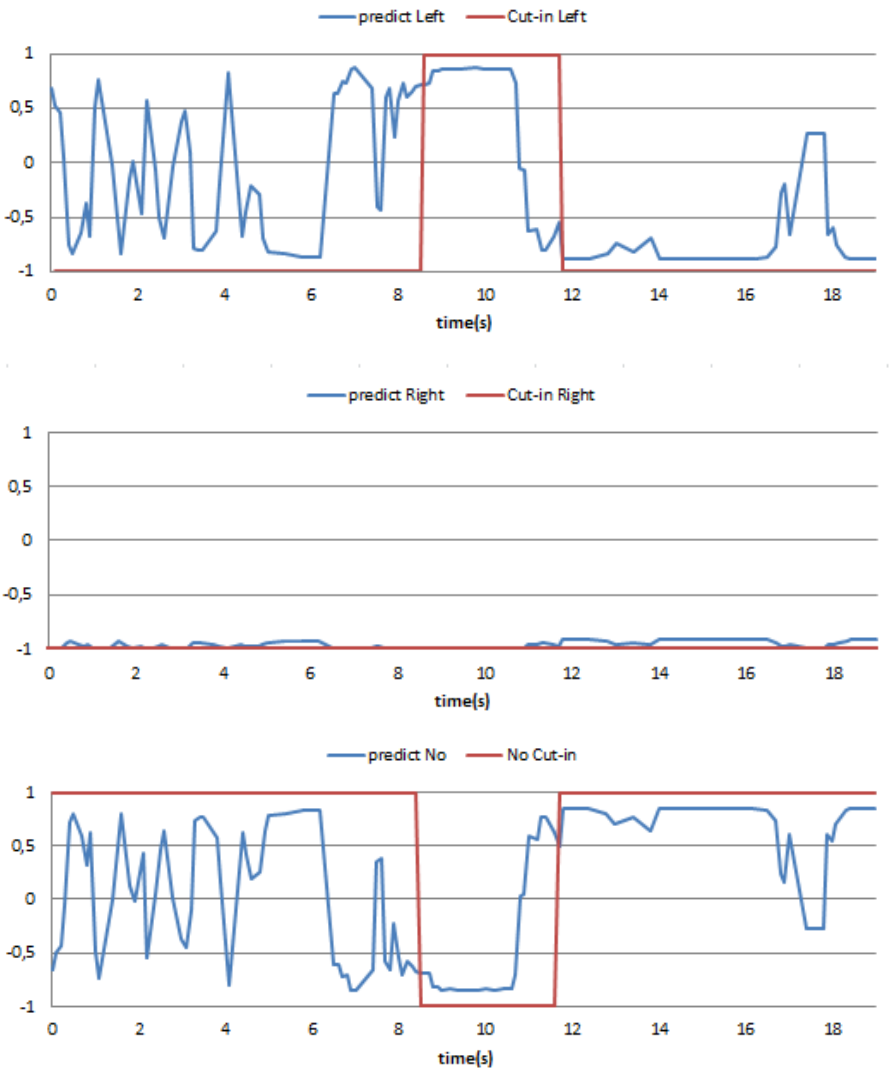


Figure 64: Prediction on a 'glued' scenario, with weights tuned on the 2 seconds prediction database.

However, the detection of the cut-in can be observed as a more consistent block starting at time $6.5s$ and ending at $10.7s$. While the ground truth defines the real cut-in between time $8.5s$ and $11.6s$. The too ambitious labelling at 2s in advance induces instabilities and difficulties to classify because of the absence of differences between examples labelled *cut-in* and examples labelled *nocut-in*.

Our algorithm is able to predict cleanly 1 second in advance. Additionally to the capacity of prediction of our algorithm, the time to classification is to take into account in our embedded application. In effect, the classifier with its weights tuned is launched every little time-step to detect in real time if there is a cut-in situation or not. Through theses examples, the time to classify a situation with the neural network is 0.1 ± 0.01 millisecond. It seems a satisfactory performance comparing the 20 milliseconds time-step in which the sensors send their signals.

13 Next step

13.1 The cut-in position estimation

Our algorithm is capable of cut-in prediction classifying the examples regarding the features extracted during the learning process that defines the discrimination between the three classes. It's obtained if there is a vehicle lane change on the right or on the left of the ego vehicle lane. But this lane change can be a cut-in, it means a lane change made very close to our ego vehicle, as a change of lane 40 meters ahead of our ego vehicle. The case in which a vehicle is doing a cut-in ahead of a vehicle which is on the ego vehicle lane ahead of our ego vehicle is also classified by our algorithm as cut-in. Finally to be able to discriminate real cut-in from change of lanes, the change of lane vehicle position estimation is necessary. Additionally, knowing the position of the danger zone is necessary to command the vehicle actuators, and define the safer trajectory and allure.

The idea of this proposition is to obtain a position estimation of the lane changing vehicle. Additionally to the labelling regarding the three classes,the solution proposed here is simply to add as output vector a 2911 cells vector that represents the cut-in vehicle occupation map where the cell containing the cut-in vehicle is putted to the high value and the others to the low value. In fact, when there is no cut-in in an instant t, this matrix is only composed by 0 values (71x41=2911 cells). When a vehicle is doing a cut-in, its position (x,y) is converted in a value 1 in the cell (red on the next figure) where its rear middle axle point is. We obtain a cut-in vehicle occupation grid for each example.

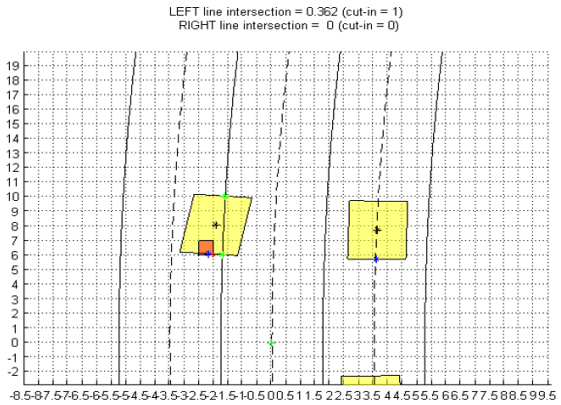


Figure 65: The *mapCut* matrix is composed of the 2911 cells representing the cut-in vehicle occupation (cut-in in red).

Adding this danger position estimation 2911 cells map to the 3 outputs variables amounts to add 2911 others classes to the 3 first. These 2911 classes give the information of presence or not of a cut-in vehicle in their associated cell. Therefore, these 2911 new classes aren't independent with the 3 original classes. When a cell of this map is activated, it means that the left or right cut-in class is activated.

Because of this add of classes, there is to reconsider a sampling of the data. In effect, there is the necessity to balance the number of examples for each 2911 classes. The idea here is to determine the maximum number of examples that represents a class (a cell), and add aleatory examples at each class (over-sampling [Bro15]) till each class is similarly represented into the database.
An adaptation of the number of the hidden neurons can be necessary to obtain a more expressive neural network, that can learn more finely the features that permit to discriminate these more restrictive classes.

This extension of our algorithm is implemented but regarding the consequent adding of parameters necessary to be able to classify theses new 2911 more classes, the number of weights to optimize is very high and the time of calculation too elevated to obtain results on our standard computer configuration.

13.2 The convolutional neural network

Convolutional neural network is a variant of multi layer perceptron neural network that we have used in this project. This kind of machine learning method obtains very good performances in the images recognition problems. They are used for example by Facebook or Google to express automatically in one sentence what is on an image. Because of our choice in working on our project with graphical 2D maps, this solution seems to be a good proposition for future trials. Therefore, this kind of method requires the use of ready to use machine learning libraries. The main objective is to optimize the classification process reducing the amount of data on its input.

- To do that, the idea is to previously extract the features using little filters (in practise a filter is a matrix of predetermined weights) that are translated through the entire input picture. If the filter has a similar pattern than the picture zone above which it is, the treatment return a high value that is stocked on a convoluted picture (the first matrix on the right of the boat image on figure 66). This convolutional process permits to obtain a feature matrix that presents the spatial positions in which there is a similar pattern to the filter. Various filters are used through the picture, and their number determines the depth of the process.

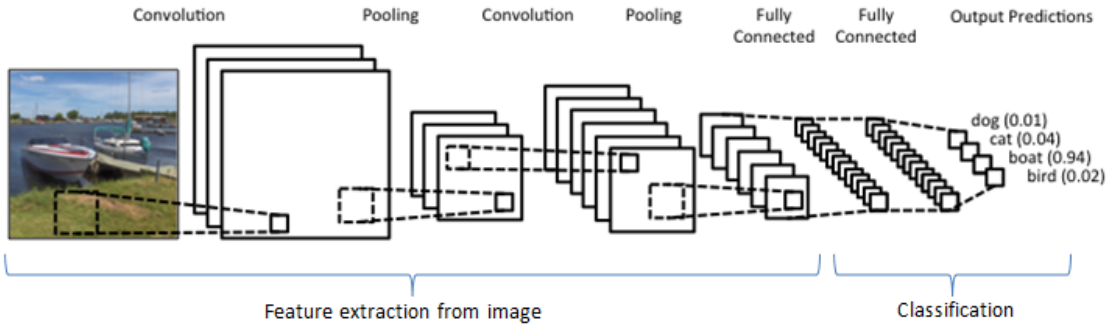


Figure 66: A convolutional neural network representation. [Inc17].

- Then it's used to simplify each feature matrix replacing each negative values by zero, introducing a non linearity with a ReLu element wise operation.

- Then comes the spatial pooling step. This step is simply a process that reduces the size of the feature matrix. The Max pooling for example consists in defining a spatial neighbourhood (for example, a 2×2 window), and to conserve only the maximum value. Doing that, we conserve only one element, and the feature matrix size is reduced conserving only the most important information.

Theses steps can be repeated several times to reduce the data to the minimum conserving only the most valuable information. The last step is the classification step. This classification step is done by a multi layer perceptron architecture as we have worked on on our project. From the reduced features maps, we find the characterizations of each classes. Extended explanations on the whole convolutional neural network can be found in [Kar16].
Regarding our application, this solution would reduce the time of calculation during the learning phase, and would permit to obtain a spatial non variability in the detection of cut-in characterization.

13.3 The multi layer perceptron without 2D maps representations

One other proposition is to let besides the graphical representation researched in our project to reduce the amount of parameters conserving the inputs (position, speeds, accelerations) explained by their normalized amplitude. In effect we have proposed the occupation grid solution to be able to fix a number of inputs to our neural network, independently of the number of vehicles around our ego vehicle. In effect our actual solution permits to take into account a non limited number of vehicle in our window of view, as long as our grid discretization is thin enough.
This other approach consists to take another way : instead of discretize the position (X,Y) in a cell occupation on a 2911 cells matrix, we could take directly the X,Y coordinates (previously

normalized) in input of our neural network, the same with the speed (speedX,speedY) and the acceleration (AccelX, AccelY). So we obtain 6 inputs per vehicles. The problem of the number of algorithm inputs persists. We should set a fixed number of inputs.

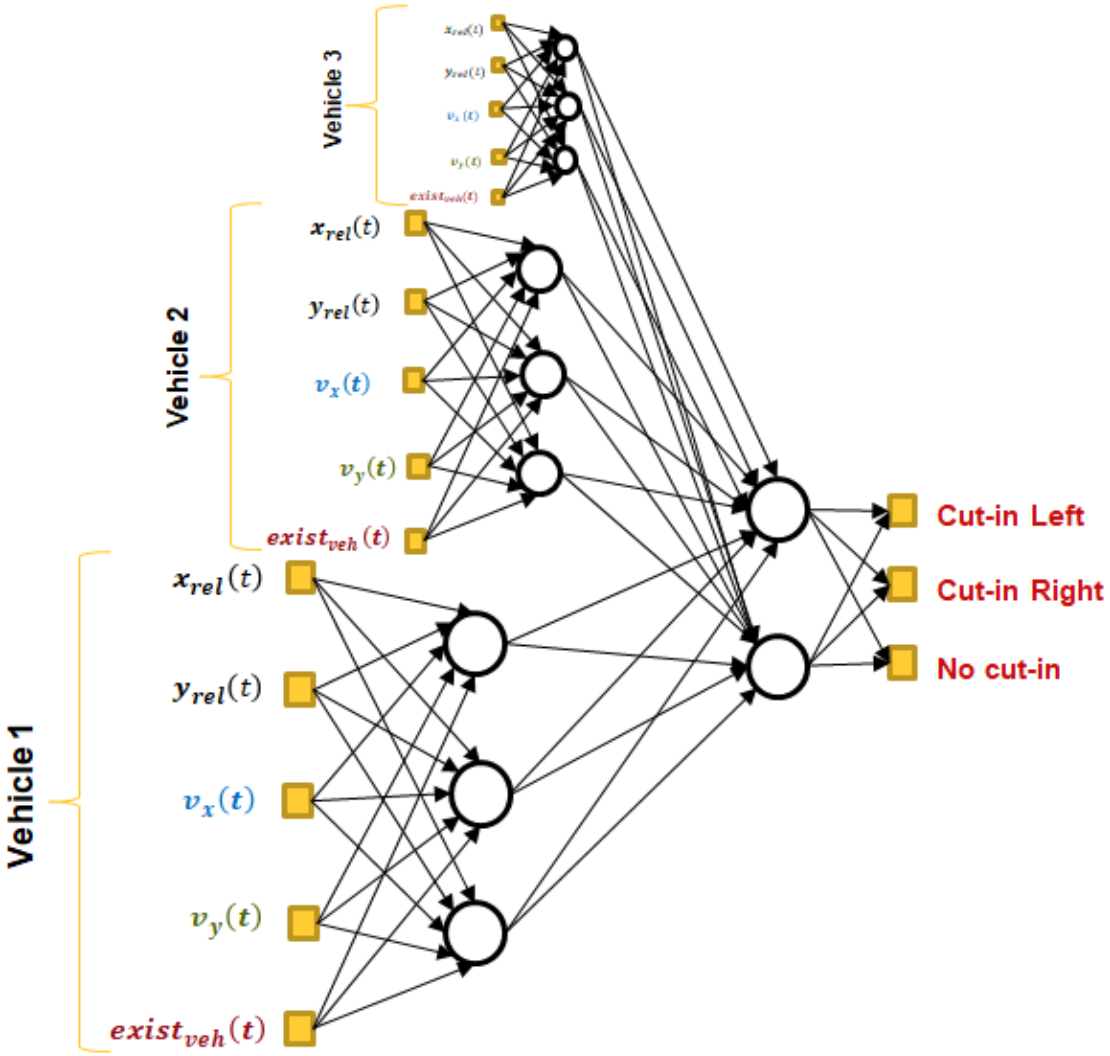


Figure 67: A graphical representation of the direct exploitation of the inputs amplitude, without the building of occupation maps. Here is proposed an architecture that takes into account the positions and speeds of three vehicles maximum. We can see the additional input '$exist_veh$' that permits to keep the existence information of the vehicle on the window of interest.

One idea is to set a maximum number of vehicles taken into account on the window of view. For example if we set this maximal number to 60 vehicles, there is to take care about the fact that if we have less than 60 vehicles, we should put the 6 inputs associated to each 'non existent' vehicle to 0. However, X=0,Y=0,speedX=0,speedY=0,accelX=0 and accelY=0 means on the relative axis that this vehicle is on the origin, it means superposed on the ego vehicle or very near. For safety considerations, a seventh input can be added to each vehicle detected on the window of view. This seventh input is an input that define if the vehicle is existent or not (1 if this vehicle is detected,-1 if not).

This solution should simplify calculations, but presents a lack of robustness. In effect, if one of the data is not valid, there is no redundancy, and the situation can be bad classified. With our solution with occupations 2D maps, a vehicle can be detected on several cells that represent the discretized area of the vehicle detected by the sensors.

14 Conclusions

The main objective of this project was to propose a way to predict cut-in scenarios for ADAS and autonomous vehicles, taking into account the surrounding vehicles.

We have developed an automated simulator and build a database that try to represents the main changes of lanes scenarios that can occur on the road. We have introduced variability, on the trajectories with wobbling trajectories generations, randomized variations of speed of each vehicles but also randomized the radius of curvature of the road. We have implemented a multi perceptron layer neural network, and done choices in order to complete real time and robustness considerations:
The choice of simplifying at the maximum the algorithm, choosing the minimal number of hidden neurons possible for the application, inducing more robust behaviour.
The choice of taking into account only the instantaneous data to predict the future, regarding the number of data to treat and the complexity, but also the assessment that this short time evolution information of trajectories is contained into the speeds and accelerations.
The choice of putting in map form the data, to obtain a kind of physical interpretation through the visualization of the 2D weights maps, that allows a certain level of verification.

We have proven with this work that there is an interesting path to develop : methodologies that observe the entire context around the ego vehicle that allow to extend the anticipation capacity for the decision cell. In effect, these kind of methodology permits to extract the interactions between the vehicles, that are sources of lane changes or dangerous trajectories. As the human instinct, and the road experience of the driver, instead of looking at the consequence (cut-in) induced by a context (positions and speeds of various vehicles), the idea is to look directly at the context that induces the consequences. That way, the consequences occurrence can be assessed and the anticipation can be done with a relatively good prediction time, with a very interesting time to decision.

Future works would treat the optimization of the code (using for example libraries) and the use of big calculators to reduce the time of calculation constraint in off-line and optimize the embedded code in on-line. This would permit to reach better classifications of the examples, so more precise extraction of features, and better stability during the classification. It would be interesting to train this type of algorithm on a real database from prototype recordings, containing more varied type of scenarios, and see the behaviour of it facing more realistic data.
Additionally to the propositions done before, one other possibility for the future is to take directly in input of the neural network the whole sensors signals without previous fusion data treatment. The algorithm would be able to extract the data of interest through the noisy data. However, this 'black box' treatment raises doubts, the engineer having no more tuning control over the physics of the system. The advantage of this kind of algorithm would being the speed of calculation.

References

[Bro15] Jason Brownlee. 8 tactics to combat imbalanced classes in your machine learning dataset. August 2015. http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7.

[cd16] RENAULT communication departement. Tech'off n13, l'aube du vehicule autonome. *Physical Review*, november 2016.

[con15] Management concil. Véhicule autonome v 1.2. Technical report, july 2015. http://www.pfa-auto.fr/wp-content/uploads/2016/03/Objectifs-de-recherche-Vehicule-Autonome.pdf.

[HON01] Antti HONKELA. Multilayer perceptrons. *Website*, may 2001. http://neuralnetworksanddeeplearning.com/index.html.

[ICK08] Ignacio ICKE. Overfitting. *Website*, February 2008. https://en.wikipedia.org/wiki/Overfitting.

[Inc17] Clarifai Inc. How does clarifai's visual recognition api work? 2017. https://www.clarifai.com/technology.

[Jac13] Lee Jacobson. Introduction to artificial neural networks - part 1. December 2013. http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7.

[Kar16] Ujjwal Karn. An intuitive explanation of convolutional neural networks. August 2016. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[Mes15] Imre Meszaros. Highly automated test solutions for camera sensors. Robert Bosch GmbH, june 2015. http://www.ukintpress-conferences.com/uploads/SPAVTDC15/d1_s2_p3_imre_meszaros.pdf.

[Ng11] Andrew Ng. Stanford machine learning. *Notes*, 2011.

[NIE17] Michael A. NIELSEN. Neural networks and deep learning. *Website*, january 2017. http://neuralnetworksanddeeplearning.com/index.html.

[RT.16] RT.com. France greenlights driverless car trials on public roads. *Website*, august 2016. https://www.rt.com/news/354683-france-driverless-car-trials/.

[XG10] Yoshua BENGIO Xavier GLOROT. Understanding the difficulty of training deep feedforward neural networks. 2010. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[YL98] Christopher J.C.BURGES Yann LECUN, Corinna CORTES. The mnist database. *Website*, 1998. http://yann.lecun.com/exdb/mnist/.

15 Appendix

GROUPE
RENAULT

SCRIPT DOCUMENTATION

Neural network & Database conception for cut-in prediction

Emilien LAURET

September 1, 2017

Abstract

This document is the explanation of my 6 months internship work in the data fusion Renault team. I explain the main work of the developed scripts and how to use them. There are two parts.

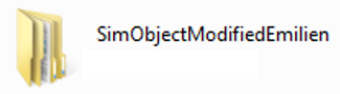
The first part deal with the simulated database generation thanks to the automation of SimObject simulator. I will explain the different steps to obtain an appropriated database for the learning phase of the Neural network algorithm.

The second part deal with the explanation of the Neural network code that I have developed, and how to train it on the generated database. It is more precisely a Multi Layer Perceptron composed by one hidden layer and the output layer.

Contents

1 The database conception

Relative scripts referring to the database conception code is contained in the *SimObjectModifiedEmilien* folder.



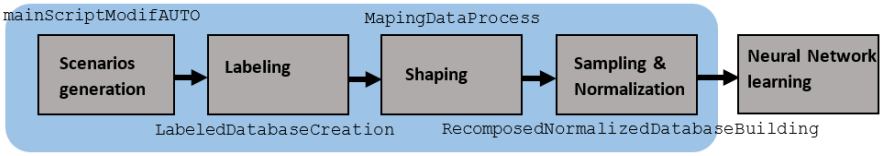


Figure 1: The various steps (and scripts) for the database conception.

1.1 Scenario definition and generation

`mainScriptModifAUTO` is the modified simulator script that automatize the generation of scenarios. Line 31 can be defined the number of scenario we want to generate:

```
for it=1:10
```

Three main cases are defined in our study:

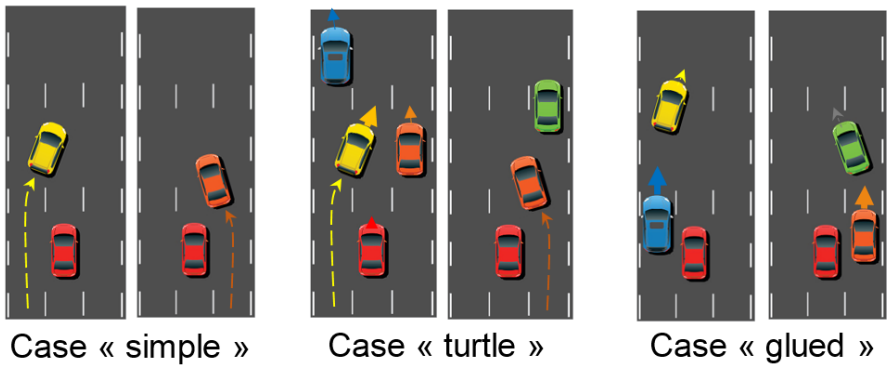


Figure 2: The three different cases I defined in the examples generation.

Each scenario duration is 30 seconds and contains one cut-in. Line 40 is randomly defining the current scenario case:

```
caseScenario = rand(1);
caseNothing=0;caseCutIn=0;caseTurtle=0;caseGlued=0;
if caseScenario>1 % nothing special
    caseNothing=1;
elseif caseScenario>0.66 % right & left cut-in ( bad behaviour & and ...
    ↪ insertion lane )
    caseCutIn=1;
elseif caseScenario>0.33 % low vehicule induce left cut-in per "zig ...
    ↪ zag" vehicule
```

```
        caseTurtle=1;
    else % high speed vehicule follow very closely an other vehicle ...
        ↪ passing our ego vehicle -> possible short cut-in
        caseGlued=1;
    end
```

The caseNothing is discarded because it produces without cut-in scenarios,that is an over represented class. The three others cases cover sufficiently the generation of no cut-in cases. The call of `ScenarioDefinition` script line 52 defines, assigns, from the kind of scenario defined previously (caseNothing, caseCutIn, caseTurtle, caseGlued) the entire parameters defining the scenario (road geometry, init position of the vehicles, speeds):

```
ScenarioDefinition
```

We can see graphically on the next figure some variable parameters that introduce the variability on the database:

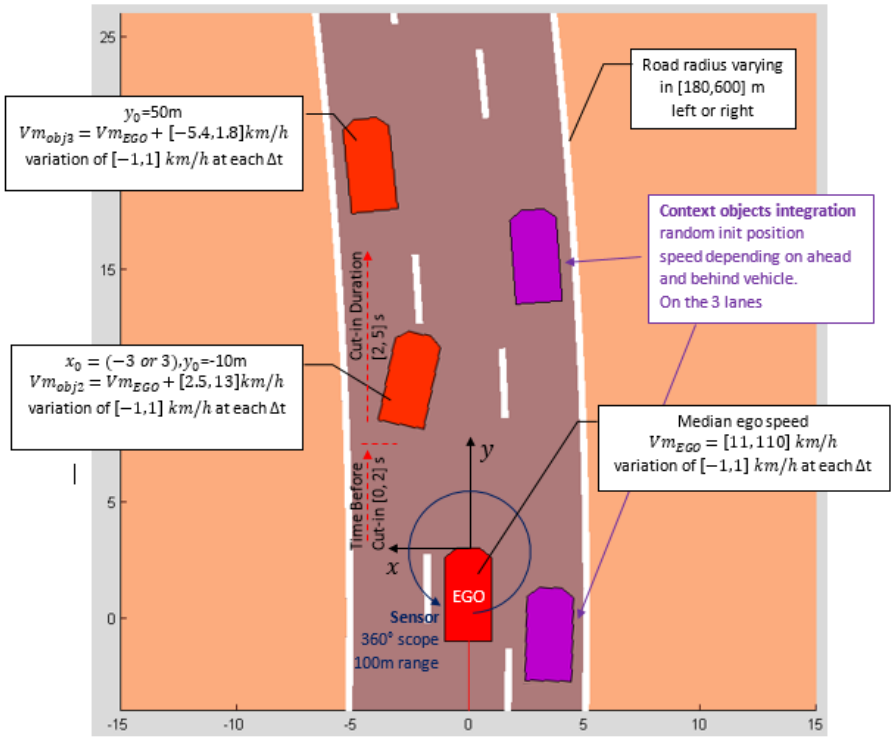


Figure 3: The various variant parameters that gives variability to the database.

There is a specific manipulation to do to obtain a convenient variation of the "time to do the lane change". In effect, practically we have experienced that changing the `changeDurations` internal parameter of the original SimObject simulator do not change consequently the "violence" of the lane change. I have defined on a first trial this variable as the following example `changeDurations = randi([2,8])` line 108 on `ScenarioDefinition.m` but the changes still stay violent. To generate scenarios with smoother lane changes, we have to manually go to the `computeTrajectory.m` original script from the simulator, and divide the alphaRef angle by 3 for example.

```
        %/////////////changing the violency of the changes/randi([1,8])
        alphaRef = atan(K * lateralError / v)/3;

        DeltaRef = (psiRef + alphaRef);
```

Theses angles *alphaRef* and *deltaRef* are the command angles that define the way to follow the predefined trajectory.

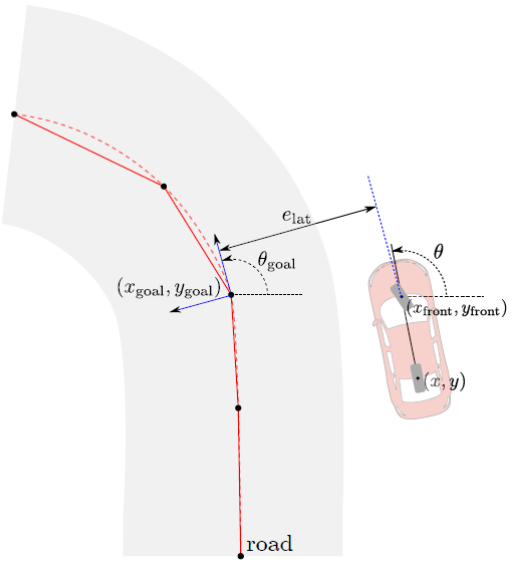


Figure 4: A graphical representation of theses angles extracted from the Description of SimObject [Bra16].

The main idea is that *deltaRef* is the angle that permits to follow the road curve. It tends to be the same value than the road's curvature. Meanwhile the *alphaRef* is the angle that add the angle's command that permits to go on the defined path, and reduce the static error e_{lat}.

Then the `insertionContextVehicles` is launched line 54:

```
insertionContextVehicles
```

This script takes the init positions and speeds of the two (for caseNothing, caseCutIn) or three vehicles (for caseTurtle, caseGlued) defining the scenario case, and randomly introduces context vehicles where they don't induce crashes. You can chose the number of context vehicle line 7 of `insertionContextVehicles` script:

```
MaxNumberOfVehPerLane=20;
```

That means it will integrate randomly a maximum of 20 vehicles per lane.

From this main definition, the simulator is launched from line 62:

```
sim = Simulator(data.roadDef, data.objectsDef, data.egoDef, data.sensorsDef);
```

A window appears and asks for the type of sensor to be used for simulation. In SimObjectModifiedEmilien \src\scenarios you can find various kind of sensor parametrization. In our case, we use a 360° angle of view sensor with constant 100 meters scope, defined by `SCENARIO_fullVision100`. If fact, we want to generate the simulated sensors fusion data that gives us the entire context data around the vehicle, without obstructions. The timestep for data generation is 0,001s.

I added an anti-collision generation part of code: It can occurs that because of some approximations in the position evolution of the objects (random variation of speeds), some vehicles come in crash to each other. This part of code comes after the simulation results, looks at the $sim.collisionVeh$ variable that determines if there are two vehicles overlapping, and suppress all the involved vehicles. Then, the IDs of each objects are updated (to obtain a continuity) and the simulator is relaunched (from line 66):

```
%% suppression of possible collisions
if sim.collisionVeh==1 % collision occured in simulation
    % delete vehicles concerned in collisions
          fprintf(['Number of objects: ...
              ↪ ',num2str(length(data.objectsDef)),'\n']);
          disp('Collision occured...suppression of impliqued ...
              ↪ vehicles...////////////////////////');
          nbCollisionVeh=length(sim.collisionVehNames);
          for column=1:size(sim.collisionVehNames,2)
              fprintf(['Object ...
                  ↪ ',num2str(sim.collisionVehNames(2,column)),' ...
                  ↪ deleted\n']);
          end
          data.objectsDef(sim.collisionVehNames(2,:))=[];
          fprintf(['Number of objects: ...
              ↪ ',num2str(length(data.objectsDef)),'\n']);

       % rename Objects after deleting elements
       for n=1:length(data.objectsDef)
           data.objectsDef{1,n}.name=['object',num2str(n)];
       end

       % create the simulator
sim = Simulator(data.roadDef, data.objectsDef, data.egoDef, ...
    ↪ data.sensorsDef);
simSettings.displayMode = 'driver';%'birdeyeNear';%'birdeyeFar';%

end
```

The simulation is launched, and it will display "Simulation loop finished" when it's finished. The results are saved in SimObjectModifiedEmilien \src\results\SCENARIO_fullVision100.

1.2 Labeling

From these data, we need to classify each example in the class he belongs. `LabeledDatabaseConception` is the script that associates the class to each example. It takes in input the previous data from SimObjectModifiedEmilien\src\results\SCENARIO_fullVision100, more precisely the `relativeGT` data of each scenario that gives objects data from the ego vehicle origin and without taking into account obstructions (vehicle hidden by another one). The shape of this kind of data is an array of 42 columns representing various data as x, y, V_x, V_y, a_x, a_y and a number of lines permitting that, for each time step, and for each object in the scope of the sensor to have one line dedicated.

Timestamp	ObjId	Xr	XrStd	Yr	YrStd	VXr	VXrStd	VYr	VYrStd	...			
0	2	-9,14	0	3,41	0	2,05	0	0,23	0				
0	3	6,57	0	-3,54	0	-0,58	0	-0,09	0				
0	4	5,70	0	3,46	0	2,68	0	-0,11	0				
0	5	17,54	0	3,19	0	3,53	0	-0,42	0				
0,001	2	-9,13	0	3,41	0	2,05	0	0,22	0				
0,001	3	6,57	0	-3,54	0	-0,58	0	-0,09	0				
0,001	4	5,71	0	3,46	0	2,68	0	-0,11	0				
0,001	5	17,54	0	3,19	0	3,53	0	-0,42	0				

Figure 5: The relativeGT data format from the simulator.

From this shape, we keep only the relevant data selecting the columns of interest line 49:

```
index=[1:2 3:2:21 33,34 35:2:41];
```

Additionally, we do not want a precision of 0,001s. We decide to conserve a time step of 0.1s for our application (line 54), for mater of reduction of data treatment:

```
for tst=0:0.1:max(tracksDataMod(:,1))
tracksData=[tracksData ; ...
    ↪ tracksDataMod(find(round(tracksDataMod(:,1)*1000)==round(tst*1000)),: ...
    ↪ ) ];
end
```

The stage of labeling consists in determining no cut-in, cut-in left or cut-in right using the overlap function.

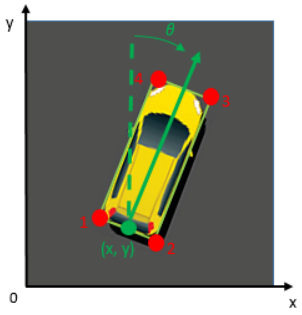


Figure 6: The `fourCoins` function determines from the yaw and the gravity point of the vehicle the coordinates of the four coins of the vehicle.

Knowing the road lines position in the simulated scenarios, the idea was to automatize the labeling task. First, I developed a function that gives us the four corner coordinates from the coordinates of the center point of the rear vehicle axle and the yaw of the vehicle (line 74):

```
[bx,by,bG] = CarCoin(tracksData,tracksData(:,11),tracksData(:,12),
                     tracksData(:,9),tracksData(:,3),tracksData(:,4));
```

Secondly, I used an overlap calculation function developed by Maxime DEROME and Abderrahim AIT-AZZI. This function takes in input the radius of the road and the position of the four corners of the vehicle (line 178):

```
%intersections//////////////
  if c(1)>0
    [interLeft{a}, leftOverlap(a),cutInLeft(a)]
=computeLineIntersectionAndOverlapCercle(c,Rayon1,bx(a,:)',by(a,:)',
tracksData(a,11),tracksData(a,12), Vobj, -1,sg);
    [interRight{a}, rightOverlap(a),cutInRight(a)]
=computeLineIntersectionAndOverlapCercle( c,Rayon2,bx(a,:)',by(a,:)',
tracksData(a,11),tracksData(a,12), Vobj, 1,sg);
  else
    [interLeft{a}, leftOverlap(a),cutInLeft(a)]
=computeLineIntersectionAndOverlapCercle(c,Rayon2,bx(a,:)',by(a,:)',
tracksData(a,11),tracksData(a,12), Vobj, -1,sg);
    [interRight{a}, rightOverlap(a),cutInRight(a)]
=computeLineIntersectionAndOverlapCercle( c,Rayon1,bx(a,:)',by(a,:)',
tracksData(a,11),tracksData(a,12), Vobj, 1,sg);
  end
```

Remember we defined the road as a constant radius circular one. It gives us in output if it is a right, left cut-in or no cut-in situation, and the overlap associated (% of object area into the ego vehicle lane). The % of overlap defining the cut-in can be modified line 199.

```
%BUILD INDEX VECTOR ////////////////////////////////
  threasholdCutIn=0.25;
```

If we want to classify the examples with predictive capacity, the labeling is done similarly but when a cut-in is averred, the labeling is extended back in time before the time it occurs. The choice of the predictive time for labeling can be defined line 200.

```
    timePrediction=1; % in seconds
```

The labeling step consists in concatenating the proper target vector to the input vector. From this manipulation, it's obtained the pair of input/output necessary for the neural network learning phase.

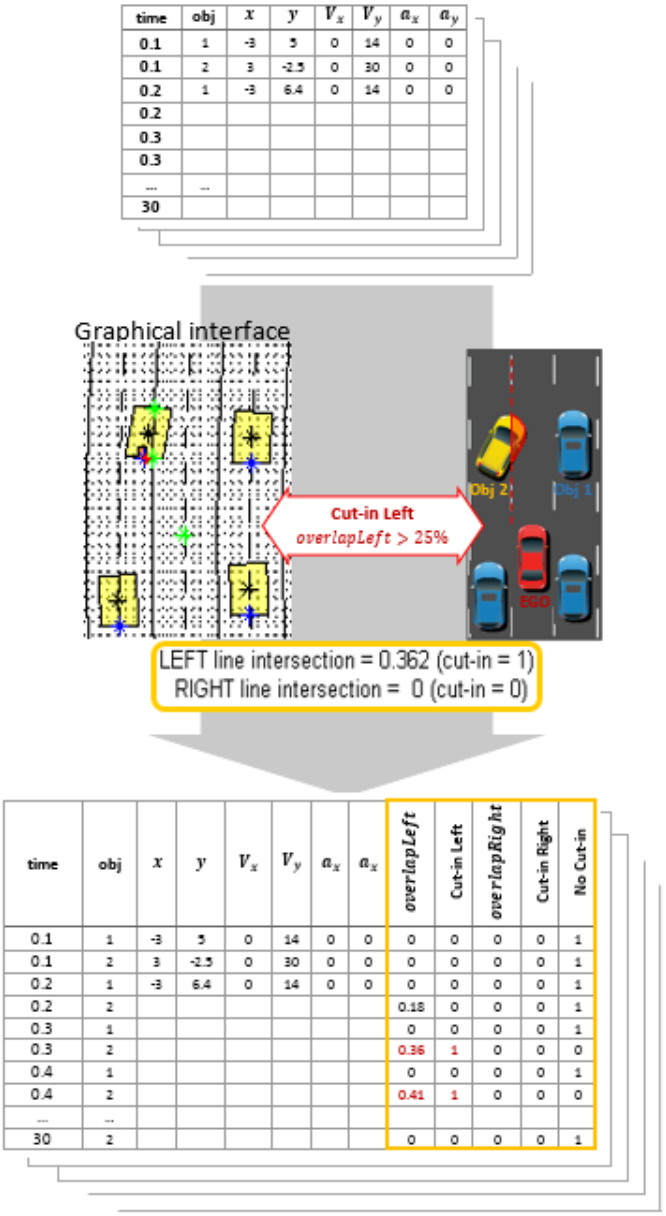


Figure 7: A graphical representation of the `LabeledDatabaseConception` script action.

A big part of the `LabeledDatabaseConception` is dedicated to create a graphical display that permits to verify the correct classification of the scenarios, and correct line intersections estimation with the associated values of overlap.

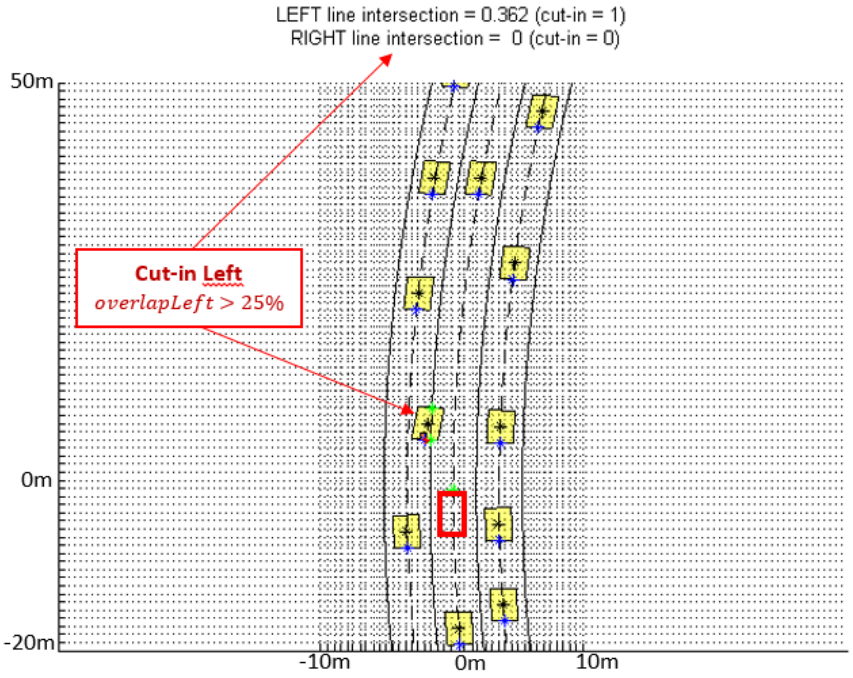


Figure 8: A view of the labeling display generated by the `LabeledDatabaseConception` script.

Additionaly to the labeling regarding the three classes, I added the generation of a matrix representing the occupation grid related to the cut-in vehicle. The grid is defined as shown figure 12, page 13. In fact, when there is no cut-in in an instant t, this matrix is only composed by 0 values (71x41=2911 cells). When a vehicle is doing a cut-in, its position (x,y) is converted in a value 1 in the cell (red on the next figure) where its rear middle axle point is. We obtain a cut-in vehicle occupation grid for each example.

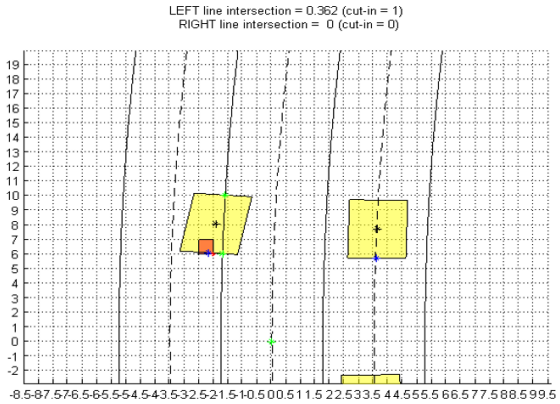


Figure 9: The *mapCut* matrix is composed of the 2911 cells representing the cut-in vehicle occupation (cut-in in red).

Line 135 can be find the position estimation grid definition for this labeling :

```
%MAP DEFINITION and initialization
        mapStepy=0.5;
        mapStepx=1;
        minYMap=-10; maxYMap=10;
        minXMap=-20; maxXMap=50;
        vectMapY=[minYMap:mapStepy:maxYMap];
```

```
        vectMapX=[minXMap:mapStepx:maxXMap];
    %input MAP DEFINITION for object position and design the network
    %architecture.
        mapStepyInput=0.5;
        mapStepxInput=1;
        minYMapInput=-10; maxYMapInput=10;
        minXMapInput=-20; maxXMapInput=50;
        vectMapYInput=[minYMapInput:mapStepyInput:maxYMapInput];
        vectMapXInput=[minXMapInput:mapStepxInput:maxXMapInput];
```

I developed a function that gives in output an occupation matrix (cell=1 when there is a vehicle and 0 otherwise) from the vehicle position (x,y) and the definition of the grid:

```
function [mapCutA]=GraphicMap(position,vectMapY,vectMapX,maxYMap,maxXMap)
```

This function is used to create the occupation map for the present labeling process. It's assigned on the next code, for each example of each scenario at each time and for each vehicle the cell number associated to its position on the grid:

```
if tracksData(a,3)≤maxXMap && tracksData(a,3)≥minXMap && ...
    ↪ tracksData(a,4)≤maxYMap && tracksData(a,4)≥minYMap

    position=[tracksData(a,3) tracksData(a,4)];
    mapObjPosition(:,:,a)=GraphicMap(position,vectMapYInput,vectMapXInput,
    maxYMapInput,maxXMapInput);
    inputNb=find(mapObjPosition(:,:,a)==1);
    tracksDataWindow=[tracksDataWindow; tracksData(a,:) inputNb];
```

This cell number `inputNb` is stored on the last column of the `tracksDataWindow` matrix and will be used in the mapping process. Finally from this process, the two maps defined previously have to be the same to have the correspondence of the matrix's `inputNb` coordinate.

The main format of the labeling output is composed of three parts : the `tracksDataWindow` that contains the entire vehicles data presents in our defined window range, the `indexation` vector that contains the binary classification in the three classes and the overlap information associated at each `tracksDataWindow` time. Finally the `mapCut` matrix is the occupation grid for the cut-in vehicle. `colnameindexation` and `colnametracksDataWindow` contain respectively the column names of `indexation` and `tracksDataWindow`.

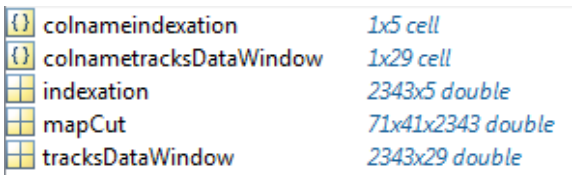


Figure 10: The outputs of the labeling step.

The results are saved in SimObjectModifiedEmilien\src\database\1000_fullData if the time for predictive labeling is chosen equal to 1s, in SimObjectModifiedEmilien\src\database\2000_fullData if 2s, SimObjectModifiedEmilien\src\database\0_fullData if 0s.

1.3 Shaping data in maps

The original idea of our project is to treat data as maps. So there is the necessity to convert ours data vectors into matrix shape representing the 2D surroundings. `MapingDataProcess.m` shapes labeled data in matrix as it can be seen on the next figure:

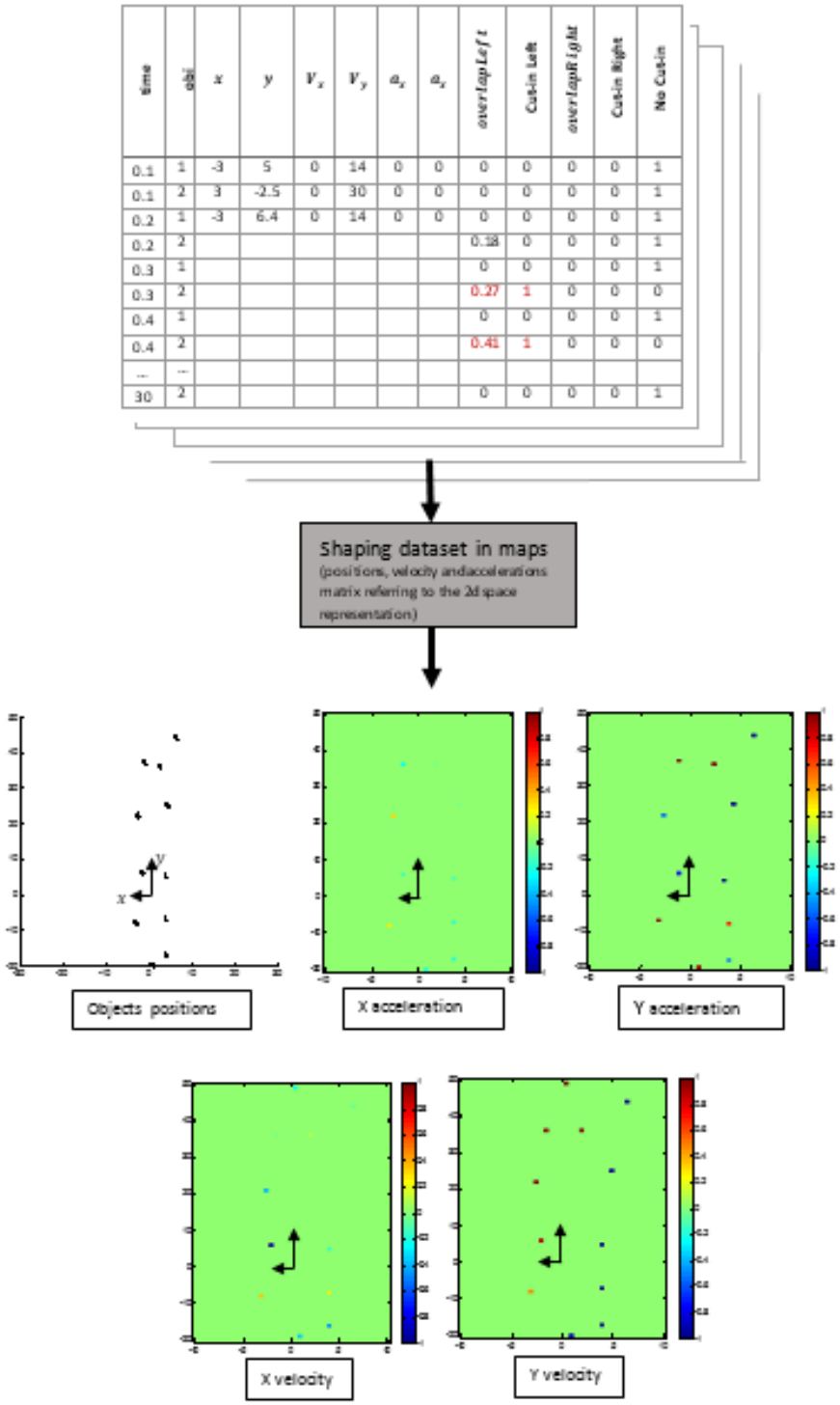


Figure 11: A view of the shaping done by the `MapingDataProcess` script.

The first lines of the MapingDataProcess.m is the definition of the grid. We have done the choice to discretize the ego vehicle surrounding (window of 30m ahead, 20 behind, and 10 on the both sides) in $1 * 0.5$ meter cells, which gives us a 2911 cells map: $(\frac{50+20}{1} + 1)(\frac{10+10}{0.5} + 1) = 2911$.

We add one cell on lateral and longitudinal to take into account the position of the ego vehicle. We care about objects positions, speeds both longitudinal and lateral, and accelerations both longitudinal and lateral. Shaping theses 5 matrix in one vector, we obtain a $2911 * 5 = 14555$ lines vector.

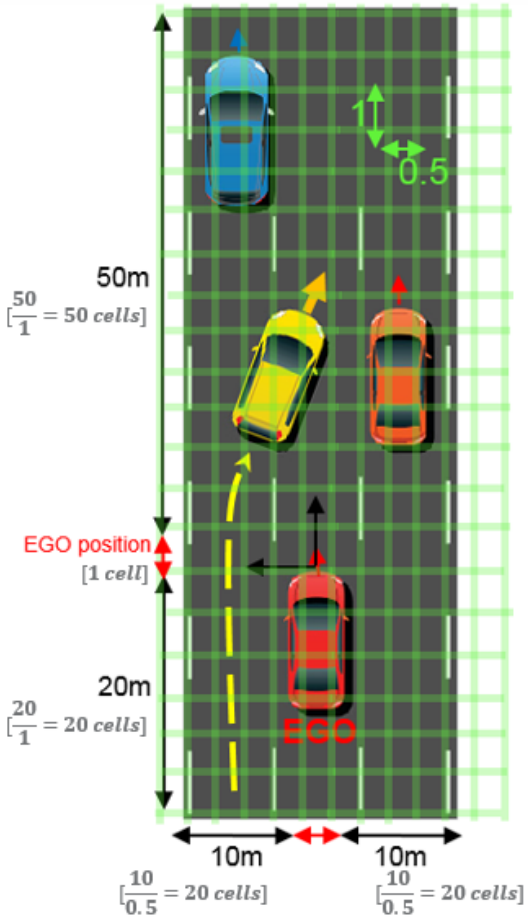


Figure 12: The definition of the grid.

From line 1 of the `MapingDataProcess.m` we define the grids for the position estimation output (label for the cut-in vehicle position estimation, so the 2911 TargetVector elements after the 3 firsts elements defining the 3 classes) target and the grid for the input data form (it's finally the same grid than for the labeling position estimation process:

```
%% MAP DEFINITION
        mapStepyInput=0.5;
        mapStepxInput=1;
        minYMapInput=-10; maxYMapInput=10;
        minXMapInput=-20; maxXMapInput=50;
        vectMapYInput=[minYMapInput:mapStepyInput:maxYMapInput];
        vectMapXInput=[minXMapInput:mapStepxInput:maxXMapInput];
```

The `tracksDataWindow` contains the list of each present vehicle in the defined window for each timestep. Using the `inputNb` column 29 of the `tracksDataWindow` matrix added in the labeling process, and given in input of the `MapingDataProcess.m`, a rapid occupation grid can be generated at a 0,1s timestep. Just putting the cells corresponding to the `inputNb` at the value 1:

```
%Creation of vehicles' positions map for each timestep
        cellsT=tracksDataWindow(find(round(tracksDataWindow(:,1)*100)
        ==round(t*100))',29);% find 'inputNbforNN' corresponding on the ...
            ↪ time t
        mapObj=zeros(length(vectMapXInput),length(vectMapYInput));
        mapObj(cellsT')=1;
```

The column 5 of the `tracksDataWindow` contains the X speed values of each vehicle. The column 6 is the Y speed. So, for each instant t, we select each vehicles represented on the window, the `mapSpeedX` and `mapSpeedY` take on the cell number `inputNb` of each vehicle the value corresponding to each vehicle speed (line 78) :

```
%Creation of vehicles' speed map for each timestep
        mapSpeedX=0*ones(length(vectMapXInput),length(vectMapYInput));
        %0.5->0km/h , 0-> min speed(negative) , 1-> max speed
        mapSpeedY=0*ones(length(vectMapXInput),length(vectMapYInput));
        mapSpeedX(cellsT')=tracksDataWindow
            (find(round(tracksDataWindow(:,1)*100)==round(t*100))',5);
        mapSpeedY(cellsT')=tracksDataWindow
            (find(round(tracksDataWindow(:,1)*100)==round(t*100))',6);
        mapSpeedsX(:,:,aT)=mapSpeedX;
        mapSpeedsY(:,:,aT)=mapSpeedY;
```

To simplify, we put in map all the velocities values on the cells corresponding to each detected vehicle position in our window. The lasts lines of this part of code is only the shaping of data in a 3D matrix, corresponding in one 2D matrix for each time `aT`. It's done the same with the accelerations line 92:

```
%Creation of vehicles' acceleration map for each timestep
        mapAccelX=0*ones(length(vectMapXInput),length(vectMapYInput));
        mapAccelY=0*ones(length(vectMapXInput),length(vectMapYInput));
        mapAccelX(cellsT')=tracksDataWindow
            (find(round(tracksDataWindow(:,1)*100)==round(t*100))',7);
        mapAccelY(cellsT')=tracksDataWindow
            (find(round(tracksDataWindow(:,1)*100)==round(t*100))',8);
        mapAccelsX(:,:,aT)=mapAccelX;
        mapAccelsY(:,:,aT)=mapAccelY;
```

An occupation grid to coordinates (x, y) conversion function was developed to be able to display the matrix `mapCut`.

```
function []=objCutPlotMap(mapCut,a,mapStepy,mapStepx,maxYMap,maxXMap,color)
```

To overcome the size of the matrix, we have split in various files. So the entire data is divided in parts containing each one 50 scenarios. In effect, regarding the number of examples labeled obtained in SimObjectModifiedEmilien\src\database\1000, we can divide this number by 50 and modify line 17 the number of parts necessary to cover the amount desired of examples:

```
1 for part=1:8
```

For example : if we have generated 400 scenarios, we will obtain $\frac{400}{50} = 8$ parts generated. We can see line 49 the separation in 50 scenarios packages:

```
for scenariosDatabasetoTest=1+(part-1)*50:50+(part-1)*50;
```

Each part covering 50 scenarios finally obtained is composed by two main results: The Input-Matrix that each column is the concatenation of the five matrix in a vector form for each example, (each column of the 5 matrix correspond to one example) as it can be seen line 198:

```
InputMatrix=[mapObjsDATA;
             mapSpeedsXDATA;
             mapSpeedsYDATA;
             mapAccelsXDATA;
             mapAccelsYDATA
             ];
```

The TargetValuesMatrix that is the concatenation of the classification index (cut-in left, cut-in right, no cut-in) and the cut-in vehicle position indexation matrix on a column line. The format is the same as the InputMatrix, one column per example:

```
TargetValuesMatrix=[indexationDATA;
                    mapCutTDATA
                    ];
```

The results are saved in various parts as defined line 17 in SimObjectModifiedEmilien\src\database\, each part containing the equivalent of 50 scenarios.

1.4 Sampling and normalization

The `RecomposedNormalizedDatabaseBuilding.m` script is composed of two parts. The first part is the under sampling process. It consists in reducing the number of "No cut-in" examples to obtain a similar number of examples for the three classes cut-in left, cut-in right and no cut-in. We determine line 13 the approximated number of examples we want for each class (we take the maximum number of examples between the cut-in left and right class):

```
desiredNbPerClass=max(size(find(TargetValuesMatrix(2,:)==1),2),
                      size(find(TargetValuesMatrix(4,:)==1),2));
```

And then select randomly this number of examples through the entire no cut-in examples of the database:

```
noCutInCases=find(TargetValuesMatrix(5,:)==1); %find the no cut-in examples

elementsExtractedFromBigClass
=randi(size(noCutInCases,2),[1,desiredNbPerClass]);
% select randomly

noCutInCases=noCutInCases(elementsExtractedFromBigClass);
```

Line 17 the "list" of examples conserved in the database:

```
elementsConserved
=[noCutInCases find(TargetValuesMatrix(2,:)==1)
               find(TargetValuesMatrix(4,:)==1)];
```

For each matrix of 50 scenarios, we reduce the number of no cut-in frames. Because of this high reduction of the number of examples, it is now possible to put all the examples on a same matrix. Each part representing 50 scenarios see their `InputMatrix` and `TargetValuesMatrix` extracted and concatenated with all the others parts (line 47):

```
for part=1:17
    InputMatrix = [InputMatrix eval(['InputMatrix' num2str(part)])];
    TargetValuesMatrix = [TargetValuesMatrix eval(['TargetValuesMatrix' ...
        ↪ num2str(part)])];
% InputMatrix = InputMatrixFULL
% TargetValuesMatrix = TargetValuesMatrixFULL;
    toDeleteInput=['InputMatrix' num2str(part)];
    eval(['clear ' toDeleteInput ';']);
    toDeleteTarget=['TargetValuesMatrix' num2str(part)];
    eval(['clear ' toDeleteTarget ';']);
end
```

Related to the position estimation map generation, an over-sampling process should be necessary to obtain a similar number of examples for each cell of the position estimation map.

The second part is the normalization phase. For practical issues, we normalize on a first time between 0 and 1, and I adapt the normalization between -1 and 1 when we load the data on the neural network algorithm script `Main_launchNN.m` line 11:

```
%CHANGE database from [0 1] to [-1 1]
InputMatrix=2*InputMatrix-1;
TargetValuesMatrix=2*TargetValuesMatrix-1;
```

To do the $[0, 1]$ normalization, there is the necessity, first of all, to take into account the entire database to find out the MIN and MAX values. Concretely, we want the MAX and the MIN of each variable of interest on the whole database of examples. If we want to use the sigmoid function, we transform the MIN in 0 and the MAX in 1. Something to take into account: the position information is defined by a Boolean occupation grid with values 0 or 1, so the firsts 2911 elements are already normalized. (From line 61):

```
%% NORMALIZATION
InputMatrix=single(InputMatrix);
%mapSpeedsXDATA
mapSpeedsXDATAnorm=(InputMatrix(2912:5822,:)
    -min(min(InputMatrix(2912:5822,:))))/(max(max(InputMatrix    ...
        ↪ (2912:5822,:)))-min(min(InputMatrix(2912:5822,:))));
mapSpeedsXDATAnorm=single(mapSpeedsXDATAnorm);

%mapSpeedsYDATA
mapSpeedsYDATAnorm=(InputMatrix(5823:8733,:)
    -min(min(InputMatrix(5823:8733,:))))/(max(max(InputMatrix    ...
        ↪ (5823:8733,:)))-min(min(InputMatrix(5823:8733,:))));
mapSpeedsYDATAnorm=single(mapSpeedsYDATAnorm);

%mapAccelsXDATA
mapAccelsXDATAnorm=(InputMatrix(8734:11644,:)
    -min(min(InputMatrix(8734:11644,:))))/(max(max(InputMatrix    ...
        ↪ (8734:11644,:)))-min(min(InputMatrix(8734:11644,:))));
mapAccelsXDATAnorm=single(mapAccelsXDATAnorm);

%mapAccelsYDATA
mapAccelsYDATAnorm=(InputMatrix(11645:14555,:)
    -min(min(InputMatrix(11645:14555,:))))/(max(max(InputMatrix    ...
        ↪ (11645:14555,:)))-min(min(InputMatrix(11645:14555,:))));
mapAccelsYDATAnorm=single(mapAccelsYDATAnorm);
```

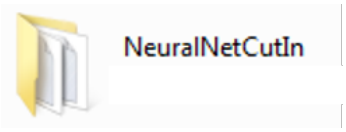
After the normalization, we concatenate each vector of 2911 elements in one vector of $2911 * 5 = 14555$ elements (line 83):

```
InputMatrix=[InputMatrix(1:2911,:);
                mapSpeedsXDATAnorm;
                mapSpeedsYDATAnorm;
                mapAccelsXDATAnorm;
                mapAccelsYDATAnorm
                ];
```

The outputs target values are already normalized in $[0, 1]$ because of their Boolean nature. So TargetValuesMatrix stay unchanged. The resulting matrix InputMatrix and TargetValuesMatrix are saved in SimObjectModifiedEmilien\src\database\1000_recomposed_FULL_DataBase.mat that contains the entire database (alls parts have been concatenated).

2 The multi-layer perceptron script

Scripts referring to the neural network code are contained in the *NeuralNetCutIn* folder.



`Main_launchNN.m` is the main script for the neural network learning phase launching. Its main scheme is graphically represented on the next figure:

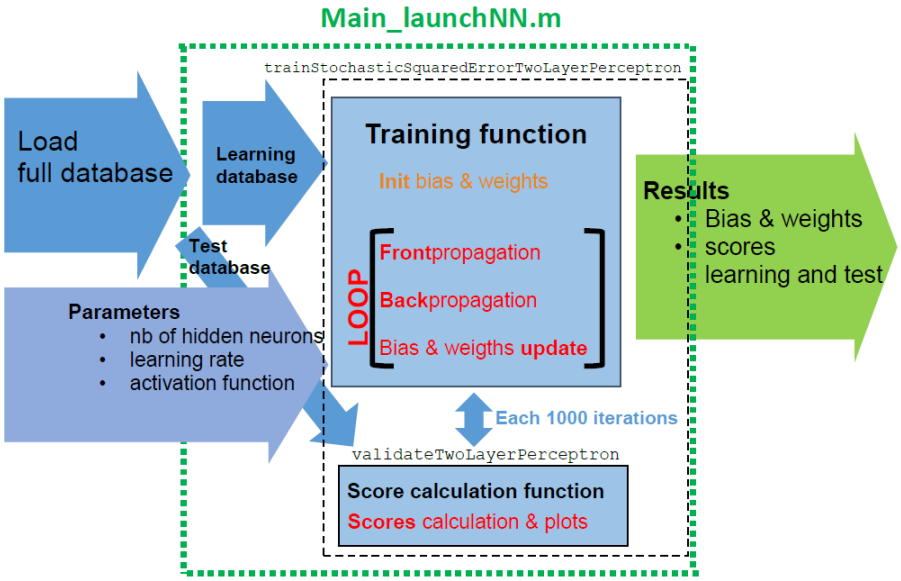


Figure 13: A graphical representation of the learning algorithm script.

There is two steps. First, load the database, from the learning and test database, define the activation function launching the `Load database` by a Ctrl+Enter (command to run one block) on the line 3 :

```
%% Load database.
addpath(genpath('d:\LocalData\p085777\Desktop\NeuralNetCutIn'));
close all; clear all; clc;

load('CUTIN_DATABASE\0_IndexedDataBase_part1.mat');
...
fprintf('Database loaded.');
```

Wait the load. The main database loaded line 7 of the *MainlaunchNN*

```
load('CUTIN_DATABASE\0_IndexedDataBase_part1.mat');
```

is divided on a learning database composed by [inputValues targetValues] vectors and a test base by [inputValuesTest targetValuesTest] vectors.

To compose these two database from the main database, it is selected randomly 10% of the examples to compose the test database, and the rest compose the learning database:

```
percentOfDatabase=1;
percentOfDataForTest=0.1; % percentage [0 to 1] of database keept to ...
    ↪ create a test base.
trainToTest=(1-percentOfDataForTest)*size(InputMatrix,2)*percentOfDatabase;
TargetValuesMatrixx = TargetValuesMatrix;
clear TargetValuesMatrix
TargetValuesMatrixx(1:2:3,:)=[]; % suppression of overlap information

% creation of the Learning and Test database
desiredNbTest
=round(percentOfDataForTest*size(InputMatrix,2)*percentOfDatabase);
elementsExtractedForTest=randi(size(InputMatrix,2),[1,desiredNbTest]);

inputValuesTest=InputMatrix(:,elementsExtractedForTest);
targetValuesTest=TargetValuesMatrixx(1:3,elementsExtractedForTest);

InputMatrix(:,[elementsExtractedForTest]) = [];
TargetValuesMatrixx(:,[elementsExtractedForTest]) = [];

inputValues = InputMatrix(:,:);
targetValues = TargetValuesMatrixx(1:3,:);
clear InputMatrix TargetValuesMatrixx
```

Line 55 is defined the activation function of each neurons and its derivative:

```
% Choose activation function.
activationFunction = @hyperbolicTangent;
dActivationFunction = @dhyperbolicTangent;
```

Then comes the second step :The launch of the learning phase. Do not launch the entire `Main_launchNN.m`, the script contains various parts of debugging and verification only used to verify the data form.

On the training function part line 89, it can be determined the main parameters: line 92 we can determine the architecture of the neural network with the choice of the number of hidden neurons, line 94 the learning rate (amplitude of weights and bias updates):

```
%% Launch Learning algorithm - 1 Hidden Layer (Sigmoid)
%STOCHASTIC ELEMENT PER ELEMENT (Online stochastic gradient descent)
% Choose form of MLP:
numberOfHiddenUnits = 10;

learningRate =0.3/1000;
```

The neural network script takes in input a database formed by a 14555 lines input vector coupled with a 2914 lines target output vector.

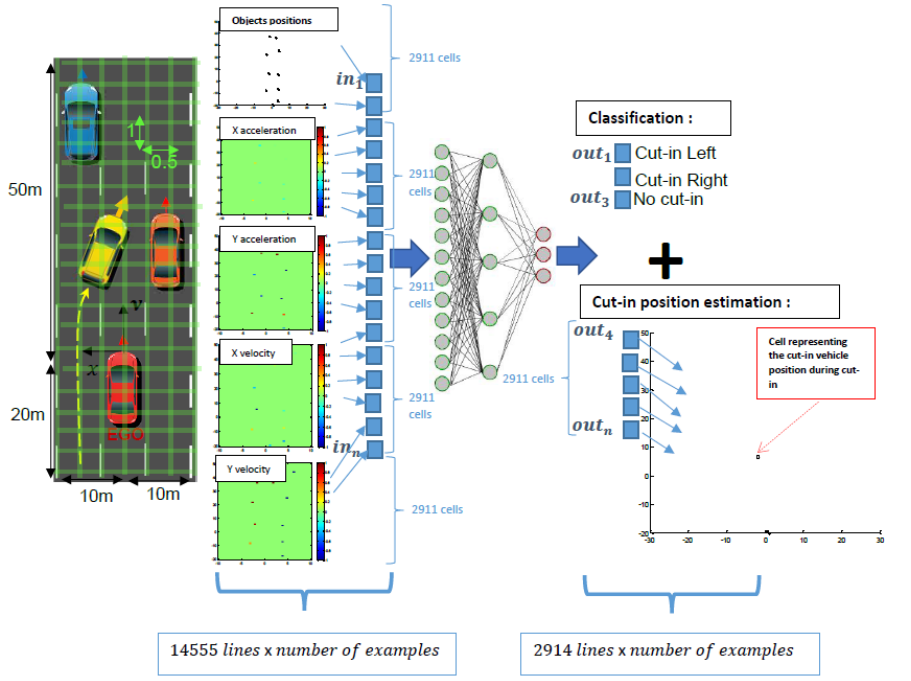


Figure 14: A graphical representation of the inputs/outputs of the neural network algorithm.

The output vector has 2914 lines. The first 3 lines are the classification in the 3 main classes: Cut-in left, Cut-in right, No cut-in. The last 2911 lines represent the map revealing the cut-in vehicle position. When a vehicle is doing a cut-in, the cell containing its rear axle middle point is raised to the high value (from -1 to 1).

So if it's only needed to classify the examples, we can keep only the three firsts lines of each examples. As the target value contains also the map For only classification learning (3 outputs), there is to restrict the outputs target vectors on theirs three first lines as follow (line 25 to 29):

```
inputValues = InputMatrix(:,1:trainToTest); % [ 14555(2911x5) x 30000 ]
targetValues = TargetValuesMatrixx(1:3,1:trainToTest);% [ 2914(3+2911) x ...
    ↪ 30000 ]
 %We keep a part of database for testing
inputValuesTest = InputMatrix(:,trainToTest+1:end*percentOfDatabase); % [ ...
    ↪ 784 x 60000 ]
targetValuesTest = ...
    ↪ TargetValuesMatrixx(1:3,trainToTest+1:end*percentOfDatabase);
```

For classification and cut-in position estimation (3+2911 outputs), let all the lines of the targets vectors:

```
inputValues = InputMatrix(:,1:trainToTest); % [ 14555(2911x5) x 30000 ]
targetValues = TargetValuesMatrixx(:,1:trainToTest);% [ 2914(3+2911) x ...
    ↪ 30000 ]
 %We keep a part of database for testing
inputValuesTest = InputMatrix(:,trainToTest+1:end*percentOfDatabase); % [ ...
    ↪ 784 x 60000 ]
targetValuesTest = TargetValuesMatrixx(:,trainToTest+1:end*percentOfDatabase);
```

We will use the stochastic gradient descent function defined in `trainStochasticSquaredErrorTwoLayerPerceptron.m`. So we can enter in this script and modify the stop condition line 100:

```
stop=(Errorscore(k)≤0.5);
```

This line means the learning algorithm will stop the learning phase when it's reached <=0.5% of error on the learning database. Returning on the `Main_launchNN.m`, the second step is the launch of the learning phase. Ctrl+Enter on the line 89 (in the Launch Learning algorithm stochastic method) :

```
%% Launch Learning algorithm - 1 Hidden Layer (Sigmoid)
%STOCHASTIC ELEMENT PER ELEMENT (Online stochastic gradient descent)
% Choose form of MLP:
numberOfHiddenUnits = 10;
learningRate =0.3/1000;

...
[hiddenWeights, outputWeights, ...
    ↪ error,Errorscore,ErrorscoreTest,epochs,time] = ...
    ↪ trainStochasticSquaredErrorTwoLayerPerceptron(activationFunction, ...
    ↪ dActivationFunction, numberOfHiddenUnits, ...
    ↪ inputValues,inputValuesTest, ...
    ↪ targetValues,targetValuesTest,learningRate,
DecreaselearningRate,plot_frequency);
fprintf('calculation finished\n');%epochs,time]
fprintf('Time for learning (min): %d\n', time/60);
```

The algorithm is learning, three plots permits to follow the squared error evolution figure 1, the scores evolutions figure 2 and the cut-in position estimation figure 3. In the "only classification case" (with 3 lines target vector), the figure 3 is not of interest. When the stop condition is reached, the message "calculation finished" appears and gives the calculation time.

Then to save the weights and bias tuned, there is to launch the `Check Learning results & save` part with a Ctrl+Enter line 138:

```
%% Check Learning results & save
    % Train score
    [correctlyClassifiedLearning, classificationErrorsLearning] = ...
        ↪ validateTwoLayerPerceptron(activationFunction, hiddenWeights, ...
        ↪ outputWeights, inputValues, targetValues);
    fprintf('Classification errors on trainning base: %d\n', ...
        ↪ classificationErrorsLearning);
    fprintf('Correctly classified on training base: %d\n', ...
        ↪ correctlyClassifiedLearning);

    % Validation TEST database
    fprintf('Validation:\n');
    [correctlyClassified, classificationErrors] = ...
        ↪ validateTwoLayerPerceptron(activationFunction, hiddenWeights, ...
        ↪ outputWeights, inputValuesTest, targetValuesTest);
    ...

    scoreTrain=correctlyClassifiedLearning*100
    /(correctlyClassifiedLearning+classificationErrorsLearning);
```

```
    ...

    scoreTest=correctlyClassified*100
    /(correctlyClassified+classificationErrors);
    ...

   % Save results
    ...
    disp('results saved');
```

Using the `validateTwoLayerPerceptron` function developed to test the neural network, we re-calculate the scores and errors obtained with the tuned parameters, the number of well and bad classed examples and save theses parameters.The `validateTwoLayerPerceptron` function do the front-propagation through the tuned neural network presented in its input, and returns the number of good and bad classified examples for this given configuration.

References

[Bra16] Eurobios Scientific Computing Branch. Description of simobject. December 2016.